SVEUČILIŠTE U ZAGREBU
FAKULTET STROJARSTVA I BRODOGRADNJE

DIPLOMSKI RAD

Slavko Hlupić

Zagreb, 2013.

SVEUČILIŠTE U ZAGREBU
FAKULTET STROJARSTVA I BRODOGRADNJE

DIPLOMSKI RAD

Mentor: Student:

Prof. dr. sc. Bojan Jerbić, dipl. ing. Slavko Hlupić

Zagreb, 2013.

Izjavljujem da sam ovaj rad izradio samostalno koristeći stečena znanja tijekom studija i navedenu literaturu.

Zahvaljujem se mentoru prof. dr. sc. Bojanu Jerbiću što mi je dao priliku izraditi ovaj rad, te savjetima koje mi je pružio tokom izrade rada.

Zahvaljujem se asistentima mag. ing. Marku Švaci i mag. ing. Bojanu Šekoranji na nesebičnoj pomoći, savjetima i potpori tijekom izrade ovog diplomskog rada.

Zahvaljujem se i svojoj obitelji koja mi je omogućila i pružila potporu tijekom mog čitavog školovanja.

Slavko Hlupić

SVEUČILIŠTE U ZAGREBU

FAKULTET STROJARSTVA I BRODOGRADNJE

Središnje povjerenstvo za završne i diplomske ispite

Povjerenstvo za diplomske ispite studija strojarstva za smjerove:
proizvodno inženjerstvo, računalno inženjerstvo, industrijsko inženjerstvo i menadžment, inženjerstvo materijala i mehatronika i robotika

Sveučilište u Zagrebu Fakultet strojarstva i brodogradnje	
Datum	Prilog
Klasa:	
Ur.broj:	

DIPLOMSKI ZADATAK

Student: **SLAVKO HLUPIĆ** Mat. br.: 0035162772

Naslov rada na hrvatskom jeziku: **SIMULACIJA RADA ROBOTSKOG SUSTAVA U PROGRAMSKOM OKRUŽENJU OPENRAVE**

Naslov rada na engleskom jeziku: **ROBOTIC SYSTEM SIMULATION IN OPENRAVE PROGRAMMING ENVIRONMENT**

Opis zadatka:

U okviru diplomskog rada potrebno je upoznati programski paket OpenRAVE, koji omogućava testiranje, razvoj i implementaciju upravljačkih algoritama te simulaciju rada robotskih sustava. Korištenjem odgovarajućih programskih alata potrebno je kreirati dva robotska modela: FANUC LrMate 200iC 5L te Kuka LWR 4+. U virtualnom okruženju OpenRAVE modelirane robote potrebno je simulirati u neurokirurškoj primjeni po uzoru na rješenje dostupno u Laboratoriju za projektiranje izradbenih i montažnih sustava. Razvijeni sumulacijski model mora omogućiti nadziranje stanja sustava u realnom vremenu tj. virtualni prikaz rada dvaju robota.

Razvijenu programsku podršku u OpenRAVE softveru potrebno je verificirati na dostupnoj opremi u Laboratoriju za projektiranje izradbenih i montažnih sustava.

Zadatak zadan:	Rok predaje rada:	Predviđeni datumi obrane:
06. prosinca 2012.	07. veljače 2012.	13.- 15. veljače 2012.

Zadatak zadao: Predsjednik Povjerenstva:

Prof. dr. sc. Bojan Jerbić Prof. dr. sc. Franjo Cajner

SADRŽAJ

POPIS SLIKA

POPIS TABLICA

POPIS OZNAKA

Oznaka	Jedinica	Opis
ms	milisekunda	izvedena jedinica za vrijeme
kg	kilogram	SI jedinica za težinu
μm	mikrometar	izvedena jedinica za duljinu
mm	milimetar	izvedena jedinica za duljinu
Nm	Newton metar	jedinica za okretni moment
°/s	stupnjeva u sekundi	jedinica brzine zakreta
kg m^2		jedinica inercije
CAD		*Computer-aided design*, izrada modela pomoću računala
wrl		format zapisa CAD datoteka
iv		format zapisa CAD datoteka
Wi-Fi		*Wireless-Fidelity*, bežična mreža
fMRI		*Functional magnetic resonance imaging*, postupak mjerenja aktivnosti mozga
xml		*EXtensible Markup Language*, jezik za označavanje podataka
TCP/IP		grupa protokola koji služe za mrežnu komunikaciju

SAŽETAK

Ovaj rad obrađuje programski paket OpenRAVE. OpenRAVE je besplatan program koji je razvijen za simulaciju robotskih sustava prije implementacije u stvarnom sustavu. Program omogućuje realno simuliranje robota, te ima ugrađen Ikfast koji omogućuje računanje inverzne kinematike robota. To omogućuje gibanje robota pomoću vanjskih koordinata.

U prvom dijelu rada su obrađene mogućnosti OpenRAVE programa. Prezentirani su načini upravljanja, način kreiranja robota i mogućnosti programa koje su potrebne za izradu robotskog okruženja i upravljanje.

U drugom dijelu rada je sve primijenjeno na stvarni sustav u laboratoriju. Kreirano je robotsko okruženje koje postoji u laboratoriju, te je uspoređena točnost i mogućnosti upravljanja robotima u OpenRAVE-u sa stvarnim robotima.

1. UVOD

Cilj ovog diplomskog rada je istražiti mogućnosti programskog paketa OpenRAVE u svrhu primjene u robotici. OpenRAVE omogućuje simuliranje raznih manipulatora te je potrebno istražiti koje su mogućnosti upravljanja robotima. Zamisao je da se u OpenRAVE-u simulira cijeli laboratorijski sustav u kojem se nalaze roboti različitih proizvođača, te zasada nema alata koji bi simulirao cijeli sustav i prikazivao sve robote.

Program je primijenjen na dva robota različitih proizvođača koji se koristite u neurokirurškoj operaciji. OpenRAVE prikazuje rad tih robota u realnom vremenu i prikazuje njihove unutarnje i vanjske koordinate.

1.1. OpenRAVE

OpenRAVE pruža okruženje za testiranje, razvoj i implementaciju algoritama kretanja sustava kod planiranja aplikacija u stvarnom svijetu robotike. Glavni fokus je na simulaciji i analizi kinematike i geometrije vezane uz planiranja kretanja. OpenRAVE je samostalan i omogućuje lagano integriranje u postojeće sustave. Pruža mnoge alate unutar naredbenog retka za rad s robotima.

Možda najbitnija tehnologija koju pruža OpenRAVE je alat koji se zove IKFast, Robot Kinematika Compiler. Za razliku od većine inverznih kinematičkih rješenja, IKFast može analitički riješiti kinematičke jednadžbe složenog kinematičkog lanca, i generirati datoteke određenog jezika za kasniju uporabu. Krajnji rezultat su izuzetno stabilna rješenja koja se mogu brzo izvoditi (oko 5ms sa standardnim procesorima).

OpenRAVE podržava COLLADA 1.5 format za specificiranje robota i dodavanje određenih nastavaka vlastitim robotima. To je otvoreni standard za razmjenu digitalne imovine između različitih grafičkih aplikacija. Collada dokumenti koji opisuju digitalnu imovinu su XML datoteke.

Proširenja robota uključuju:

- manipulatore
- senzore
- planiranje specifičnih parametara

1.2. Neurokirurgija

Povijest liječenja stara je koliko i ljudska povijest. Koliko je danas poznato, najstariji pisani dokument posvećen ozljedama glave nastao oko 1700 godina prije Krista. Valetudinaria su prve organizirane institucije u Rimskom Carstvu za skrb o ranjenim i bolesnim borcima i gladijatorima. Premda u 4. stoljeću sveti Benedikt osniva red benediktinaca, s misijom liječenja i cijeljenja, u srednjem vijeku je većina do tada stečenih znanja zanemarena, zaboravljena i napuštena. Ne događa se mnogo toga novoga sve do 19. stoljeća.

Neurokirurgija kao neovisna disciplina razvila se na prijelazu 19. u 20. stoljeće. U Engleskoj je Sir Victor Horsley bio prvi kirurg koji se specijalizirao i potpuno posvetio neurokirurgiji, u SAD-u je to bio Harvey Cushing, učenik ne manje slavnog kirurga Williama Halsteda.

Devedesetih godina prošlog stoljeća, u neurokirurgiji, odškrinuta su vrata dvorane tajni, i otpočelo je fantastično putovanje u dotad nepoznato.

Sve do nedavno (a ponegdje još i danas) mladi se neurokirurzi za vrijeme specijalizacije uče kako u ljudskom mozgu postoje aktivni centri (npr. za govor, pisanje, pokretanje lijeve ruke ili desne noge) povezani živčanim putovima. Ovaj je koncept zvučao logično i pouzdano i činilo nam se da ima nedvojbenu praktičnu vrijednost, sve dok se nedavno nisu umiješali neurobiolozi i upotrebljavajući argumente koje je bilo nemoguće osporiti, dokazali kako nijeme zone u ljudskom mozgu ne postoje. Ne postoji ni jedan djelić živčanog tkiva koji je "nijem" i beskoristan i koji može, bez ikakvih štetnih posljedica, biti rezan, koaguliran, odstranjen, potiskivan ili manipuliran na neki drugi način.

Prije bilo kakva manipuliranja živčanim tkivom potrebno je identificirati i precizno lokalizirati, koliko je to god moguće, poziciju "elokventnih zona", odnosno centara čiju smo lokalizaciju do danas donekle upoznali. Od brojnih novih tehnoloških inovacija, najviše rezultata za sada pruža predoperativna funkcionalna magnetna rezonancija (fMRI). S obzirom da je mozak „elokventna" cjelina, sve operaciju treba planirati tako da budu minimalno invazivne i time što je više moguće pošteđene živčane strukture.

Malo gdje je točna vizualizacija i milimetarski precizna lokalizacija patološkoga procesa važna za uspješno liječenje i moguće iscjeljenje kao u neurokirurgiji.

Svatko tko je sudjelovao u neurokirurškoj operaciji zna kako su i najbolji neuroanatomski atlasi i najkvalitetnije neuroradiološke snimke veoma daleko od realnoga svijeta u koji ulazimo nakon otvaranja dure. Veoma je teško orijentirati se u trodimenzionalnom prostoru. Dovoljno je pogriješiti samo nekoliko milimetara pri odabiru kuta kojim prodiremo u dubinu da bismo potpuno promašili patološki proces. [1]

2. OREMA ROBOTSKOG NEUROKIRURŠKOG SUSTAVA

U ovome poglavlju opisana je oprema koja se koristi u neurokirurškom sustavu. U tu opremu ulaze roboti sa upravljačkim jedinicama, te oprema i alati neophodni za uspješno obavljanje operacijskih zadaća.

Roboti koji se koriste su Fanuc LR Mate 200iC5L i KUKA LBR4, dok od alata i opreme koriste se markerska pločica, kamera s laserskim senzorom, stereovizijski sustav, kirurška bušilica, kateter, kalibracijska kugla i kalibracijska igla.

Najvažniji dio ovog sustava su roboti. Značajke industrijskih robota su:

- Nosivost [kg] – maksimalni teret koji robot može .
- Broj stupnjeva slobode gibanja – broj zglobova koji se mogu gibati neovisno jedan o drugome.
- Točnost ponavljanja [mm] – tolerancija koja govori o tome s kojom točnošću robot može ponavljati radnje, tj. koje mu je odstupanje ako ga se više puta pošalje u istu točku.
- Točnost pozicioniranja – odstupanje od programski zadane putanje, pozicije i orijentacije.
- Struktura
- Radni i kolizijski prostor
- Način upravljanja i programiranja
- Vrsta pogona
- Cijena
- Masa robota [kg] – bitna je za projektiranje sustava.
- Doseg [mm] – najudaljenija točka koju robot može dohvatiti.
- Granice kretanja [°] ili [mm] – granice unutar kojih robot može pokretati pojedini zglob.
- Maksimalna brzina [°/s] ili [mm/s] –brzina koju može postići pojedini zglob.
- Moment [Nm] – moment koji djeluje u zglobu.
- Inercija [kg m^2] – momenti inercije koji se mogu pojaviti na zglobu

2.1. Fanuc LR Mate 200iC/5L

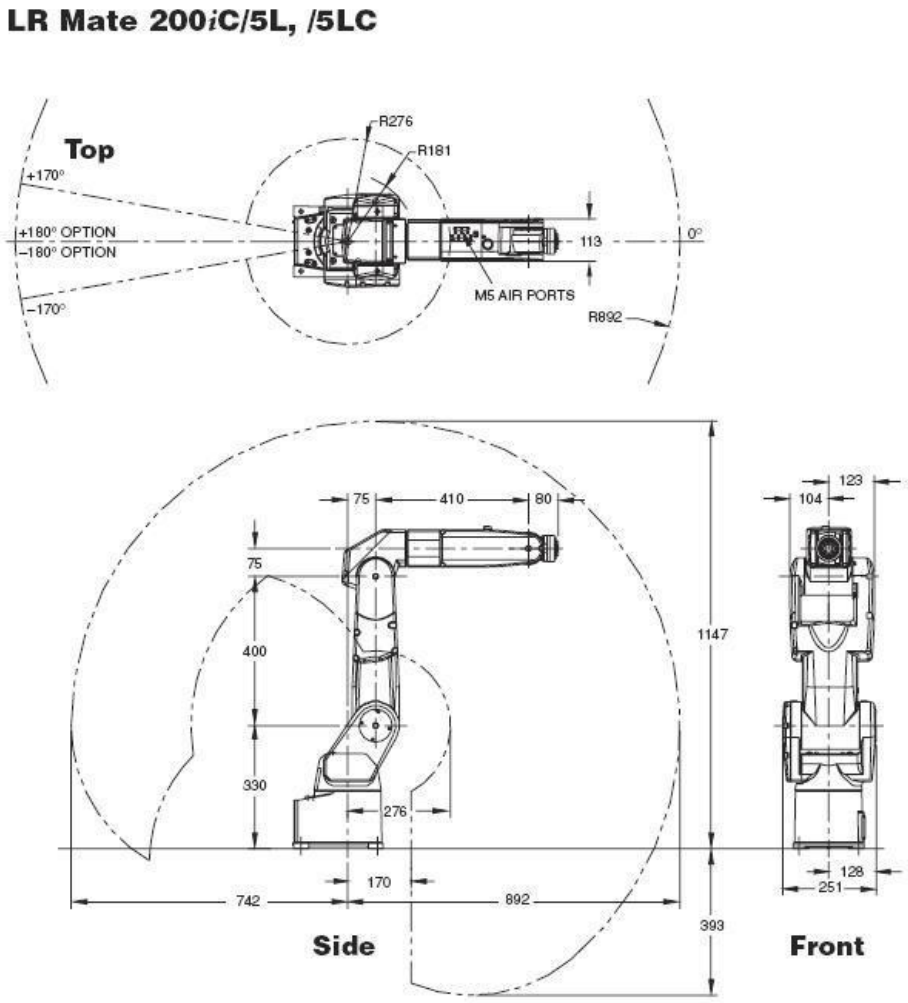
Fanuc LR MATE 200iC/5L [Slika 1] je kompaktna, visoko brzinska robotska ruka malog profila i male težine. Visoka točnost pozicioniranja i stolna veličina čine ga idealnim za razne primjene automatizacije. Integracija LR Mate 200iC i alata pojednostavljena je strateškim smještajem pneumatskih i električnih priključaka te dva elektromagnetna ventila.

LR Mate 200iC/5L roboti koriste se za razne aplikacije poput strojne obrade, rukovanja materijalima, montažom, dodavanjem i uklanjanjem materijala. Proširen doseg i tanak dizajn zglobova daje robota koji može jednostavno doći do teško pristupnih dijelova. [2]

Broj stupnjeva slobode		6
Korisna nosivost – rukavac [kg]		5
Domet [mm]		892
Ponovljivost [mm]		±0,03
Ograničenje gibanja [º]	Prva os (J1)	360
	Druga os (J2)	230
	Treća os (J3)	373
	Četvrta os (J4)	380
	Peta os (J5)	240
	Šesta os (J6)	720
Brzina gibanja [º/s]	Prva os (J1)	270
	Druga os (J2)	270
	Treća os (J3)	270
	Četvrta os (J4)	450
	Peta os (J5)	450
	Šesta os (J6)	720
Moment zgloba [Nm]	Četvrta os (J4)	11,9
	Peta os (J5)	11,9
	Šesta os (J6)	6,7
Inercija zgloba [kg m^2]	Četvrta os (J4)	0,3
	Peta os (J5)	0,3
	Šesta os (J6)	0,1
Masa robota [kg]		29

Tablica 1. Tehničke karakteristike LR Mate 200iC/5L

Slika 1. Robot Fanuc LR Mate 200iC/5L



Slika 2. Dimenzije Fanuc LR Mate 200iC/5L

2.1.1. R-30iA Mate upravljačka jedinica

Fanuc R–30iA Mate [Slika 3] je kontroler napredne i dokazane tehnologije. Napravljena je prema Fanuc-ovom modularnom konceptu, kontroler daje fleksibilnost za primjenu specifične konfiguracije. On smanjuje vrijeme ubrzanja i usporenja robota. To dovodi do smanjenja vremenskih ciklusa.

Nova poboljšanja ovog kontrolera uključuju poboljšanu kontrolu vibracija i integrirani 2D vizijski sustav koji se može nadograditi na 3D vizijski sustav. Kontroler nudi jednostavno spremanje programa na robota.

Učinkovit sustav hlađenja ima odvojene krugove hlađenja za maksimalnu učinkovitost. Stražnji protok zraka štedi prostor na podu i omogućuje postavljanje više kontrolera jedan do drugog. Također je spriječeno ulaženje prašine putem sustava hlađenja.

Značajke Fanuc R-30iA Mate kontrolera:

- Mogućnost proširenja do 40 osi upravljanja.
- Pokretanje za manje od jedne minute.
- U slučaju nestanka električne energije brzo pokretanje i nastavak programa.
- Omogućen je brz pristup svim dijelovima, te je minimaliziran broj istih.
- Nema filtra klime.
- I/O priključki omogućuju brzu promjenu alata
- Kontroler radi s Fanuc operativnim sustavom
- Kontinuiran praćenje struje servomotora zbog ranog otkrivanja grešaka.
- Otkrivanje sudara. Prati se razlika između stvarne i očekivane struje motora.
- Nadzor preopterećenja [2]

Slika 3. R-30iA Mate upravljačka jedinica

Slika 4. iPrivjesak

2.2. Kuka LBR 4+

Početkom svemirskim istraživanjima javila se potreba za robotima. Kako su roboti koji su konstruirani za svemir na Zemlji bili beskorisni, jer nisu mogli pokretati sami sebe, te za obuku astronauta javila se potreba za konstruiranjem laganog robota. S tom namjerom na DLR-ovom institutu za robotiku nastao je LWR I (Light weight Robot) [Slika 5]. Iako je razvoj potaknut za potrebe svemirskih istraživanja, LWR se probio tek primjenom u aplikacijama na Zemlji.

2000. godine predstavljen je LWR II koji je omogućavao nosivost od 7 kg, dok je robot imao masu od 18 kg. Lagana konstrukcija robota je omogućena primjenom metode konačnih elemenata u optimizaciji svih kritičnih komponenti, kao i primjenom vrlo laganih prijenosa i snažnih motora. Slično kao i ljudska ruka robot ima sedam stupnjeva slobode gibanja što mu omogućuje veću fleksibilnost.

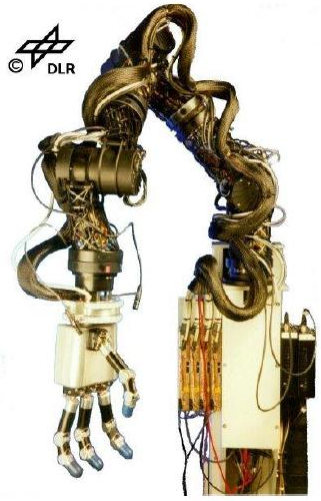
LWR III je postigao cilj o laganim robotima, te je uz masu od 13,5 kg bio u mogućnosti nositi teret od 15 kg. Specifičnost je modularna izvedba dobivena spajanjem više jednakih članaka [Slika 6]. [4]

Kuka LBR 4+ [Slika 7] je nasljednik LWR robota s DLR-a. Dok su prethodne serije laganih robota bile razvijene prvenstveno za istraživanja, LBR 4+ je projektiran za rad u industriji. Vanjska struktura robota je izrađena od aluminija, što pridonosi poboljšanju kvalitete, što je ključno za industrijsku primjenu.

LBR 4+ ima nosivost od 7 kg. Zahvaljujući integriranim senzorima, robot je odličan za rukovanje i montažu. Zahvaljujući svojoj maloj masi od 16 kg, robot je iznimno energetski učinkovit i lako prenosiv.

Ovaj robot posjeduje veoma napredni upravljački i mjerni sustav. Senzori momenta smješteni u svakom zglobu, detaljan dinamički model i brzi ciklusi upravljanja, zajedno sa snažnim motorima i laganom konstrukcijom omogućuju prigušenje vibracija s ciljem postizanja točnosti putanja. Elektronika također omogućuje vođenje robota rukom, čime se pojednostavnjuje programiranje a i omogućuje interakcija robota i ljudi. Zahvaljujući mjerenju momenata u svim osima više nije potrebna hvataljka s ugrađenim momentnim senzorima.

Na taj način može se detektirati sudar. Time je također olakšana montaža, jer robot ne mora precizno dovesti predmet, već može doći u blizinu točne pozicije i nakon toga 'tražiti' poziciju spoja. [3]



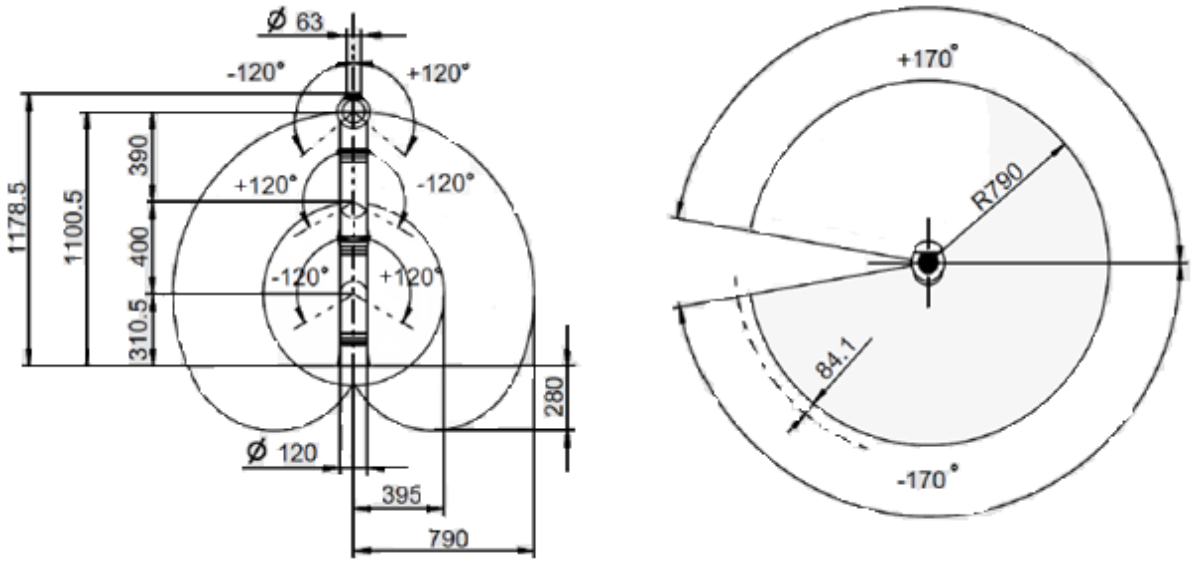
Slika 5. DLR LWR I

Slika 6. DLR LWR III

Broj stupnjeva slobode			7
Korisna nosivost – rukavac [kg]			7
Domet [mm]			1178,5
Ponovljivost [mm]			±0,05
	Ograničenje gibanja [º]	Brzina gibanja [º/s]	Moment zgloba [Nm]
Prva os (J1)	±170	110	176
Druga os (J2)	±120	110	176
Treća os (J3)	±170	128	100
Četvrta os (J4)	±120	128	100
Peta os (J5)	±170	204	100
Šesta os (J6)	±120	184	38
Sedma os (J7)	±170	184	38
Masa robota [kg]			29

Tablica 2. Tehničke karakteristike robota Kuka LBR 4+

Slika 7. Kuka LBR 4+



Slika 8. Dmenzije i radni prostor robota Kuka LBR 4+

2.2.1. KR C2 lr upravljačka jedinica

KR C2 lr [Slika 9] je upravljačka jedinica iz serije KR C2 upravljačkih jedinica. Potpuno integrirani kontrolno-upravljački koncept. Osnovna varijanta omogućava upravljanje 6 osnim robotom sa još dvije dodatne osi, a moguća je nadogradnja za upravljanje s još 4 dodatne osi. Korištenje već isprobanog i testiranog upravljačkog sistema u kombinaciji sa standardnim PC računalom osigurava veliku pouzdanost na visokoj tehnološkoj razini. Modularna struktura i upravljanje svakom osi zasebno pruža vrlo jednostavno održavanje upravljačke jedinice i laganu zamjenu dijelova. Elektronički sigurnosni koncept baziran je na sabirničkoj

tehnologiji. Koristi se odvojeno napajanje za memoriju sa dodatnim baterijama, što omogućuje spremanje pozicije u slučaju nestanka električne energije.

Ergonomski dizajnirano upravljačko sučelje KCP (KUKA Control Panel) [Slika 10] posjeduje sve potrebne funkcije, za upravljanje, kontrolu i programiranje robotskog sustava.

Slika 9. KR C2 lr upravljačka jedinica

Slika 10. KPC upravljačko sučelje

2.3. Robotski alati

Alati i senzori služe za interakciju robota sa okolišem. Pomoću senzorskih sustava robot vidi svoju okolinu, te na temelju dobivenih informacija donosi odluke o poduzimanju potrebnih radnji i/ili o korekciji putanje gibanja kako bi se izvršila određena zadaća.

Vrste senzora:

- Vizijski sustavi – kamere koje snimaju okolinu i aplikacije koje na temelju dobivenih informacija prepoznaju objekte u okolini.
- Laserski senzori – služe za određivanje udaljenosti predmeta.
- Senzori sila i momenata – registriraju sile i momente na prihvatnici.
- Ultrazvučni senzori – detektiraju objekte u prostoru i njihovu udaljenost, ali razlučivost im je mnogo manja od vizijskih sustava i laserskih senzora.

Nakon primanja informacije sa senzorskih sustava, robot je u stanju obaviti određene zadatke. Spektar alata koji se primjenjuju u robotici je velik, i u pravilu svim alatima kojima se služi čovjek može se služiti i robot; hvataljke – služe kao ruke robotu, pištolji za zavarivanje, lakiranje, poliranje, alati za obradu odvajanjem čestica i dr.

U neurokirurškom sustavu robot se može služiti svim instrumentima i opremom koju koristi i neurokirurg, uz dodatak visoko preciznih senzorskih sustava. Senzorska i kirurška oprema koja se koristi u ovim sustavom je slijedeća:

- Markerska pločica
- Kamera s laserskim senzorom
- Stereovizijski sustav
- Kirurško svrdlo
- Kateter
- Kompenzacijska kugla
- Kompenzacijska igla

2.3.1. Markerska pločica

Markerska pločica [Slika 11] služi za lociranje pacijenta prije operacije. Pločica se fiksira na lubanju pacijenta [Slika 12], te se obavi snimanje glave (CT, MRI i dr.). Na taj način se vizualizacijom mozga može točno utvrditi područje operativnoga zahvata. Na početku operacije roboti odrede poziciju markerske pločice, te znadu mjesto operativnoga zahvata.
Koordinatni sustav markerske pločice je definiran rupama na pločici. Na pločici se nalazi i metalni šiljak na vrhu kojega je kuglica koja služi za precizno pozicioniranje pločice.

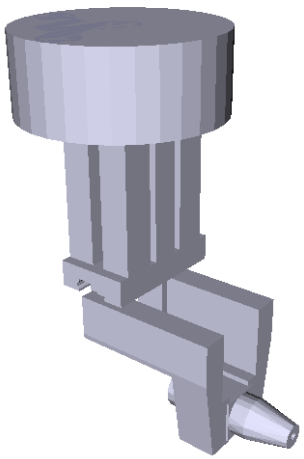
Slika 11. Markerska pločica u OpenRAVE-u

Slika 12. Markerska pločica fiksirana na lubanju

2.3.2. Vodilica

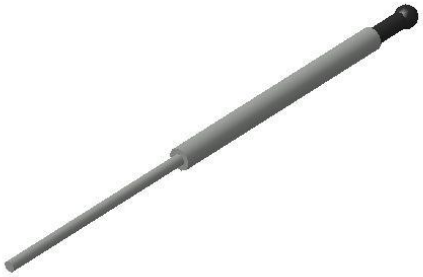
Vodilica [Slika 13] služi za vođenje svrdla i katetera u lubanju pacijenta, te služi kao hvataljka mjerne kugle.
Mjerna kugla [Slika 15] služi za precizno pozicioniranje robota u odnosu na markersku pločicu na pacijentu. Na vrhu alata nalazi se mala kugla koja je točno poznatog promjera. Robot Fanuc kamerom s laserom pronađe markersku pločicu i odredi njen položaj. Zatim robot Fanuc uzima alat vodilicu i mjernu kuglu, te se pozicionira iznad šiljka markerske

pločice. Robot Kuka tada uzima stereovizijski sustav i provjeri dali se robot s mjernom kuglom pravilno pozicionirao. Nakon tog se mjernom kuglom odredi pozicija mjerne pločice.

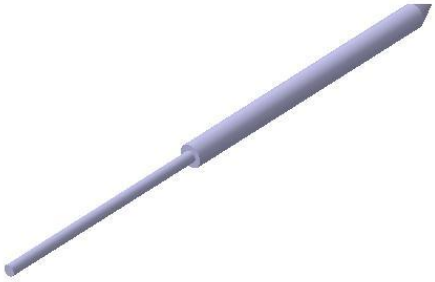


Slika 13. OpenRAVE model vodilice

Slika 14. Vodilica



Slika 15. Mjerna kugla

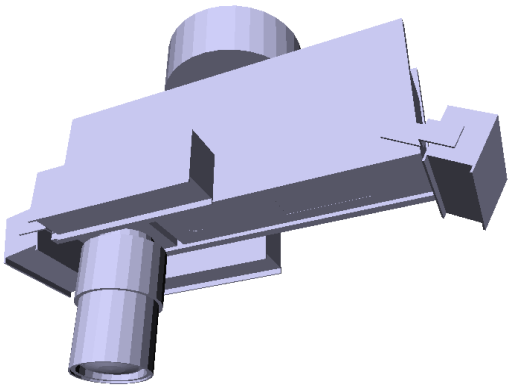


Slika 16. Mjerna igla

2.3.3. Kamera s laserskim senzorom

Ovaj alat je kombinacija kamere s laserskim senzorom [Slika 17], a njegova je svrha prepoznavanje, pozicioniranje i orijentiranje objekata u prostoru u odnosu na koordinatni sustav robota. Koristi se Baslerova industrijska kamera u boji s pripadajućim objektivom. Laserski senzor triangulacijskom metodom mjeri udaljenost objekta na način da pomoću laserske zrake stvara točku na promatranoj površini koja se pod određenim kutom promatra sa CMOS senzorom.

U kirurškoj operaciji ovaj alat se montira na Fanuc robota i služi za određivanje položaja markerske pločice.



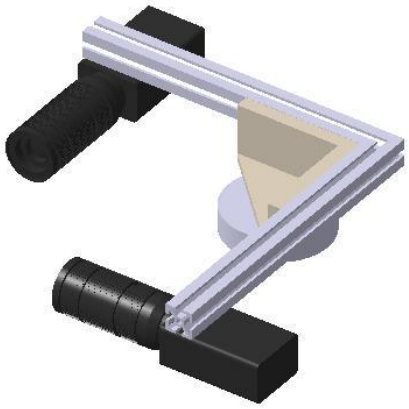
Slika 17. OpenRAVE model kamere s laserskim senzorom

Slika 18. Kamera s laserskim senzorom

2.3.4. Stereovizijski sustav

Stereovizijski sustav s najmanje dvije kamere određuje poziciju objekta u okolišu. Ovdje se koristi sustav od dvije Baslerove kamere u boji s pripadajućim objektivima [Slika 19]. Kamere su jedna u odnosu na drugu pod kutem od 90° i točno je poznata geometrija kamera, kao i njihova montaža na robotu. Na obje slike iz kamera pronalazi se ista točka, te se zahvaljujući poznavanju odnosa između kamera izračunava pozicija točke.

U neurokirurškoj operaciji ovaj alat služi za određivanje položaja markerske pločice.



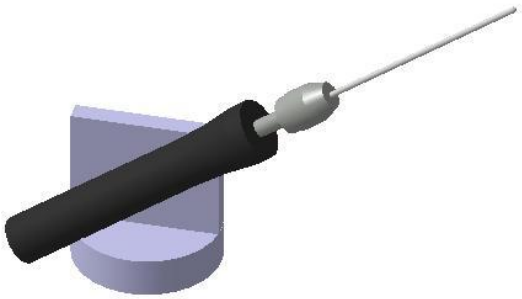
Slika 19. OpenRAVE model stereovizijskog sustava

Slika 20. Stereovizijski sustav

2.3.5. Kirurška bušilica

Kirurška bušilica [Slika 21] može biti pneumatska ili električna. Bušilice su promjenljivog broja okretaja i rade do 80000 okr/min.

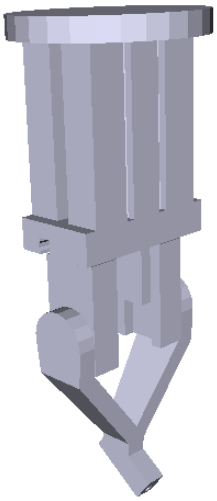
Svrdlo je od nehrđajućega čelika, promjera Ø3 mm, za rad na visokim brojevima okretaja. Površina je visoko polirana, i velike žilavosti da se spriječi pucanje.



Slika 21. OpenRAVE model kirurške bušilice

2.3.6. Kateter

Kateter [Slika 22] je cjevčica duljine 230 mm. Materijal je vrlo elastičan ali je otporan na izvijanje, tj. vrača se u prvu poziciju ako se malo savije. Impregniran je barijem kako bi bio vidljivi na rendgenskim snimkama. Na kateteru su otisnute oznake za duljinu na 5, 10 i 15 cm kako bi kirurg mogao lakše odrediti dubinu prodora. Vrh katetera je silikonski elastomer impregniran tantalom kako bi bio vidljiv na rendgenu.

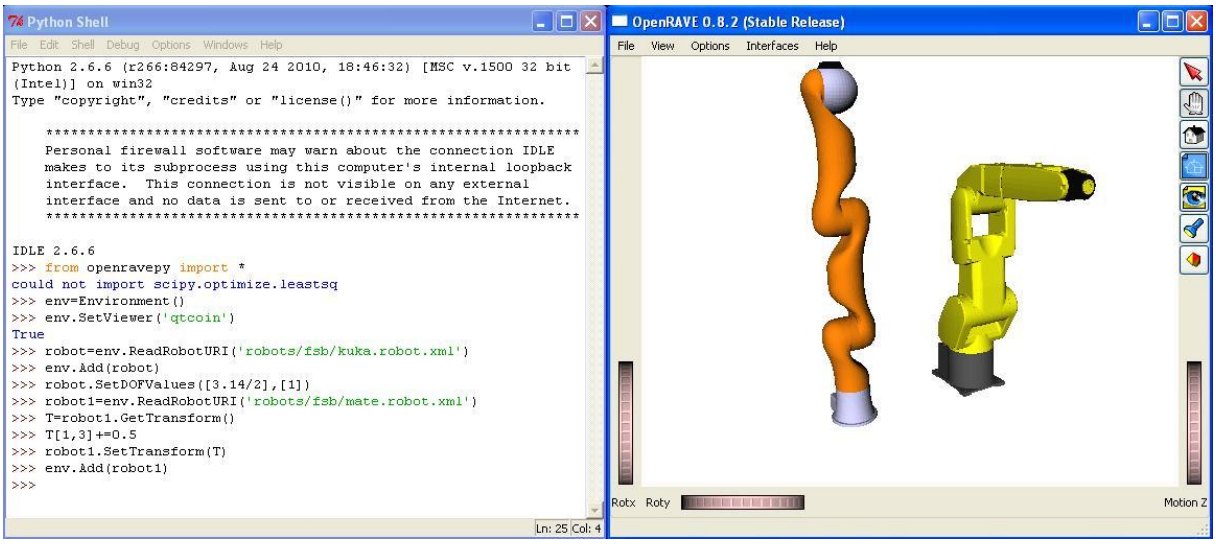


Slika 22. OpenRAVE model katetera

Slika 23. Kateter

3. OPENRAVE I KREIRANJE MANIPULATORA

Kod projektiranja automatskih sustava potrebno je detaljno ispitati planove gibanja i algoritme upravljanja. S tom namjenom je izrađen OpenRAVE, kao softver otvorene platforme. OpenRAVE je namijenjen za stvaranje okruženja robotskih sustava, a uključuje besprijekornu 3D simulaciju, vizualizaciju, planiranje i kontroliranje. Modularna arhitektura omogućuje jednostavnu izradu vlastitog kontrolera ili proširenje mogućnosti. OpenRAVE dodaci, algoritmi planiranja, kontroleri robota ili senzori mogu se distribuirati i dinamički učitavati za vrijeme izvođenja. OpenRAVE nudi rješavanje kinematike i dinamike robota, detekciju sudara i kontrolu robota. OpenRAVE arhitektura pruža fleksibilno sučelje koje se može koristiti u kombinaciji s drugim popularnim robotskim paketima kao što su Player i ROS. OpenRAVE također podržava okoliš za mrežno skriptiranje što ga čini jednostavnim za kontrolu i nadzor robota. Jedna od ključnih prednosti otvorene arhitekture je da omogućuje istraživačima robotike jednostavno dijeljenje i usporedbu algoritama.



Slika 24. Python naredbeni redak i OpenRAVE prozor

3.1. OpenRAVE

Kako mnogi istraživački napori pokazuju, stvoriti autonomnog robota zahtijeva mnogo više od modularnosti protoka podataka. Višestruke komponente visoke razine planiranja, percepcije, kontrole, nadzora pogrešaka, svi moraju besprijekorno funkcionirati u širokom rasponu ulaznih scenarija. Fokus najpopularnijih robotskih arhitektura kao ROS, Player, CLARAty i Urbi je njihova modularnost, ponovna upotrebljivost, komunikacija i

podudarnost. Dok su arhitekture, poput Carmen, Player i CLARAty, dizajnirane za planiranje navigacije, relativno malo arhitektura uključuje izravnu potporu za razvoj i testiranje algoritama planiranja koji uključuje složenu geometriju, kao što je planiranje gibanja za mobilne robote ili humanoidne robote. Stoga je predstavljen OpenRAVE kao arhitektura otvorenog koda koja se bavi povezivanjem trenutno dostupnih programa zbog svladavanje ograničenja.

Neke jednostavne značajke koje OpenRAVE čine moćnim alatom:

- Integriran dizajn za kontrolu i nadzor robota u trenutnom vremenu
- Funkcionalnost kinematičkih operacija i fizičke simulacije
- Ugrađeni su osnovni alati za planiranje manipulacije i hvatanje
- Standardni dodaci koji omogućuju testiranje različitih planskih algoritama i senzorskih sustava s minimalnim brojem modifikacija [5]

3.1.1. OpenRAVE arhitektura

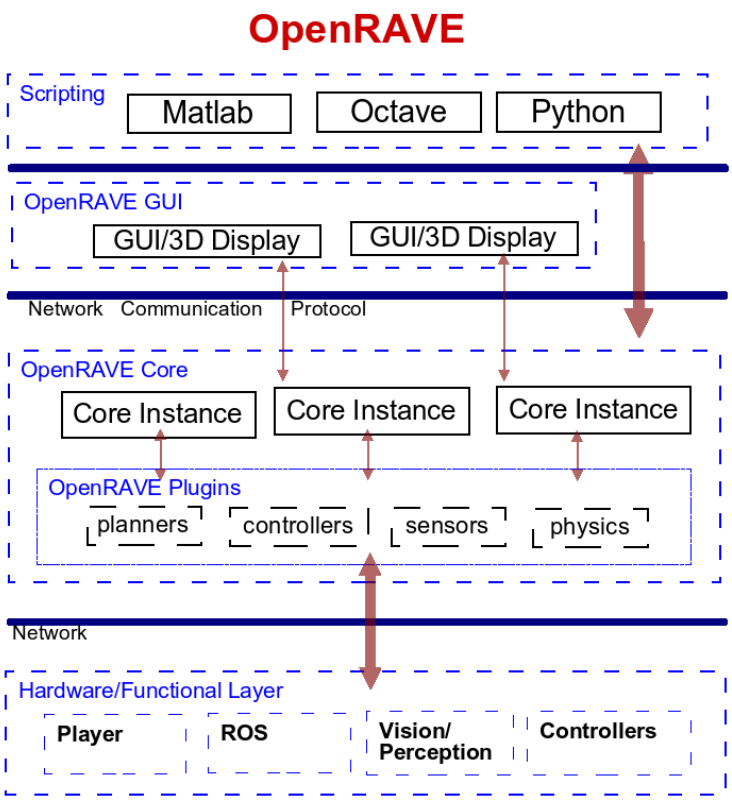
OpenRAVE modularna arhitektura, omogućuje izvršavanje i planiranje sustava u robotici, tako da razvoj autonomnih sustava postaje lakši i komponente postaju djeljive za više projekata. OpenRAVE je jedinstven po tome što pruža standardno sučelje preko planera i visoku razinu izvršavanja modula. Arhitektura je podijeljena u četiri različita sloja [Slika 25]. Sloj skriptiranja, GUI sloj, sloj OpenRAVE jezgre i sloj dodataka koji se mogu koristiti u interakciji s drugima arhitekturama robotike. Prilikom pokretanja, jezgra pokreće poslužitelja, koji upravlja ažuriranjem okoline. To omogućuje da se svi objekti trenutno učitaju i prihvaća naredbe koje mogu učitati dodatke, pozvati planere, poslati kontrolne naredbe za robote ili dodati nove objekte. GUI koristi isti klijent/poslužitelj model kao skripte, te stoga korisnik može komunicirati sa scenom kroz GUI i neprimjetno mijenjati stanje unutar OpenRAVE objekta. [5]

3.1.2. Dodaci i sučelja

OpenRAVE identificira određene kategorije sučelja koje je moguće provesti uz dodatak.
Vrste sučelja su:

- Planeri - Plan je putanja koju robot mora slijediti kako bi iz prve pozicije postigao konačno stanje poštujući određena ograničenja, kao što su održavanje dinamičke ravnoteže ili izbjegavanje sudara s preprekama.
- Controller - Svaki robot je priključen na kontroler koji se koristi za pomicanje u prostoru (simulirani ili pravi). Kontroler osigurava funkcionalnost putanje.

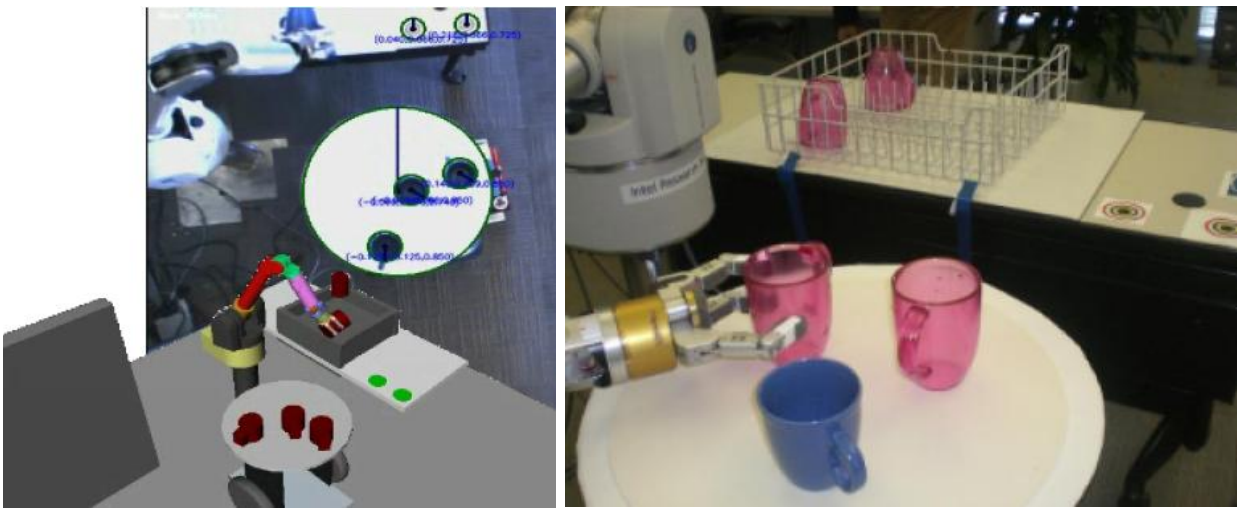
- Senzori - senzor, poput daljinomjer ili kamera, prikuplja informacije o okolišu i vraćaju u standardnom format. Senzor može biti priključen na bilo koji dio robota.
- Problem - Svaki problem je analogan malom programu ugrađenom u OpenRAVE. Jednom stvoren, problem je registriran u glavnoj simulacijskoj petlji. Problemi mogu ponuditi posebne funkcije za manipulaciju ili navigaciju i mogu proširiti mrežu sposobnosti OpenRAVE.
- Roboti - OpenRAVE podržava različite kinematičke strukture robota za s jedinstvenim sposobnostima. Na primjer, sučelje za humanoidnog robota može se znatno razlikovati od sučelja za mobilnih robota s kotačima. Pružanje implementacije za različite klase robota omogućuje klijentima da bolje iskoriste njihovu strukturu.
- Inverzni kinematički model - IK solvers može navesti i vratiti zatvorene obrasca ili numerička rješenja koja se mogu koristiti kao ulaz u planer gibanja. Svaki IK Solver može biti priključen na podskup robota. [5]



Slika 25. OpenRAVE arhitektura podijeljena u četiri sloja

3.1.3. Sučelje u realnom vremenu

Osim za simulaciju okruženja i razvoj algoritma, OpenRAVE je dizajniran kao sučelje za kontrolu robotskih sustava. Algoritmi i skripte dizajnirani za rad unutar simuliranog OpenRAVE okruženja mogu u teoriji raditi u stvarnom svijetu. U praksi je potrebno podešavanje algoritama i ispunjavanje zahtijeva za izvođenje na pravom robotu. OpenRAVE je dizajniran da pokuša minimizirati potrebu za prilagođavanjem simulacije za upotrebu u stvarnom okruženju.



Slika 26. Upravljanje u stvarnom vremenu

Primjer sa slike iznad [Slika 26] je sustav u realnom vremenu. Zadatak robota je da autonomno pokupiti šalice iz ladice montirane na Segway i stavite ih u stalak za suđe. Segway se može slobodno kretati i ladica može biti zatrpana sa šalicama u proizvoljnim položajima. Planer položaja mora uzeti u obzir mnoga potencijalna stabilna mjesta hvatanja za svaku šalicu kako bi se odlučio za najbolju strategiju. [5]

3.1.4. Manipulacija i hvatanje

OpenRAVE arhitektura može podržati različite robotske zadatke, zadržavajući njihov originalni dizajn. Inicijalni razvojni napori usmjereni su na hvatanje objekta, kao što su izračunavanje položaja objekta i krajnjeg izvršnog člana te računanje sile zatvaranja hvataljke. Većina istraživanja za hvatanje smatra da je krajnji izvršni član u mogućnosti pristupiti objektu sa svih strana. U stvarnom okruženju, izvršni elementi su pričvršćeni na manipulator čime je ograničeno kretanje izvršnog člana. Stoga, kako bi se riješio problem jednostavnog hvatanja predmeta, potrebno je uzeti u obzir krajnje djelovanje manipulatora i oblik objekta, ali i prepreke u okruženju. Problem postaje još teži kad je ruka pričvršćena na mobilnu platformu. [5]

3.1.5. Budući razvoj OpenRAVE-a

Razvoj OpenRAVE-a je još uvijek u tijeku te je plan dodati još novih značajki kako bi se omogućio lakši razvoj robota. Prvo područje je standardizacija mrežnog protokola i eventualno dijeljenje poruka s ROS-om. Drugo područje je omogućiti kontrolu za paralelno izvršavanje planera i kontrolu u realnom vremenu. Sposobnost planiranja unaprijed može povećati učinkovitost. Treće područje je eksperimentiranje s prikazom i modelima te način integracije unutar OpenRAVE. [5]

3.2. Python

Python je viši interpretirani programski jezik opće namjene čije je glavno obilježje dobra čitljivost koda. Čitljivost je postignuta inteligentnom upotrebom uvlaka koje služe za definiranje blokova koda, što se u drugim programskim jezicima postiže raznim interpunkcijskim znakovima i zagradama. Obično koristeći python možemo napraviti neku aplikaciju u znatno manje linija koda nego što bi nam bilo potrebno da smo koristili C++ ili Java-u.

Python je više-platformski programski jezik i u općem slučaju, moguće je isti program napravljen u Pythonu pokrenuti na Windows, Unix, ili OS X sustavima jednostavnim kopiranjem datoteke bez dodatnih zahvata.

Jedna od najvećih prednosti Pythona je to što dolazi sa vrlo opširnom standardnom bibliotekom koja nam omogućava da stvari poput skidanja datoteke sa interneta, raspakiravanja komprimiranih datoteka, ili kreiranja web servera možemo napraviti sa svega nekoliko linija koda.

Uz standardnu biblioteku, dostupne su tisuće drugih biblioteka od kojih neke pružaju sofisticiranije mogućnosti od standardne biblioteke poput NumPy i SymPy numeričkih biblioteka, te PyQt biblioteke za rad s Qt-om.

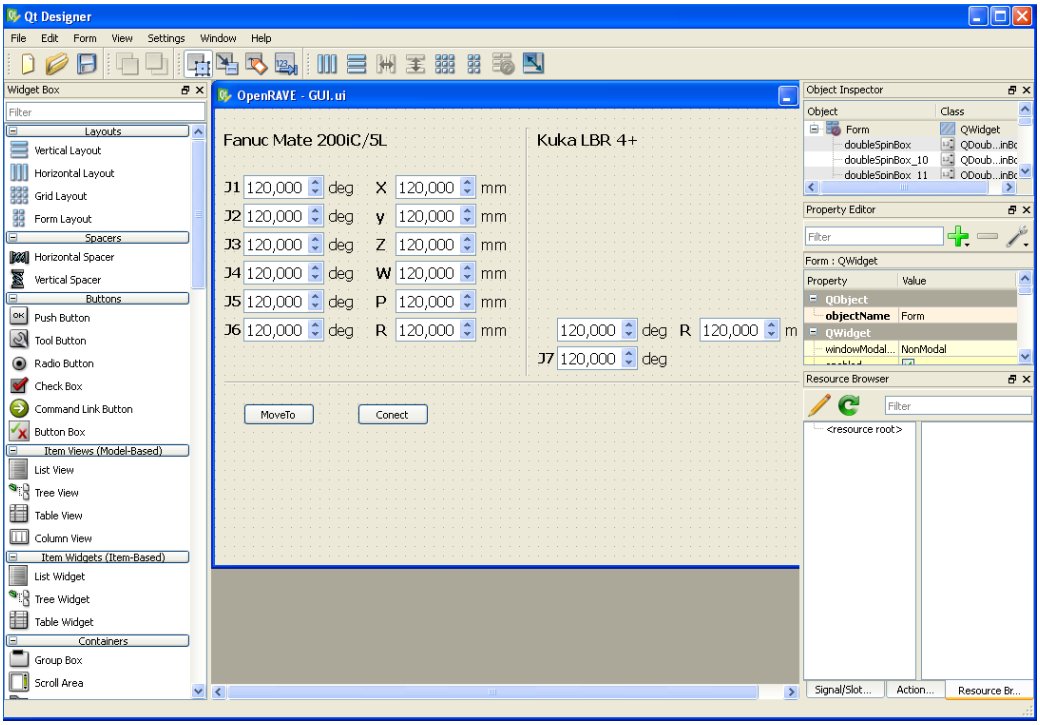
3.3. PyQt

Qt [Slika 27] je više-platformski framework za izradu aplikacija sa grafičkim korisničkim sučeljem napisan u *C++* programskom jeziku , a PyQt je njegova verzija namijenjena korištenju uz Python. Njegova glavna obilježja su upotreba *widgeta* i *signal*&slot sistema razmjene signala između pojedinih widgeta.

Kako je Qt jako velik paket, podijeljen je više modula tako da se u memoriju mogu učitati samo pojedini dijelovi paketa. Moduli su sljedeći:

- *QtCore* - Sadrži osnovne klase uključujući petlje i signal&slot mehanizam.
- *QtGui* - Sadrži klase vezane za grafička korisnička sučelja.
- *QtMultimedia* - Sadrži multimedijalne klase
- *QtNetwork* - Sadrži klase za pisanje UDP, TCP, FTP i HTTP servera i klijenata.
- *QtOpenGL* - Sadrži klase koje omogućavaju korištenje *OpenGL*-a pri renderiranju 3D grafike.
- *QtOpenVG* - Sadrži klase koje omogućavaju korištenje *OpenVG* pri crtanju
- *QtSql* - Sadrži klase koje se mogu koristiti sa besplatnim ili komercijalnim SQL bazama podataka. Sadrži promjenjive modele za tablice koji se mogu koristiti u kombinaciji sa grafičkim korisničkim sučeljem.
- *QtSvg* - Sadrži klase za prikaz sadržaja SVG datoteka.
- *QtWebKit* - Sadrži klase potrebne za renderiranje i interakciju sa web sadržajima.

Osim nabrojanih modula Qt sadrži i aplikaciju QtDesigner koja omogućuje lakšu izradu grafičkih korisničkih sučelja u stilu Visual Basica. Iako ovaj pristup omogućuje lakšu i bržu izradu kompliciranih grafičkih sučelja sa sobom nosi i određenje nedostatke. QtDesigner ne pokriva sve mogućnosti Qt-a, a da bi postigli neke stvari moramo ručno nadopisati kod.



Slika 27. Qt Designer

3.4. Promjenjivi XML format

OpenRAVE koristi XML za pohranu svih robota i opisa scena. XML je dovoljno fleksibilan za povezivanje već stvorenih objekata i robota u jednu XML datoteku. Također je moguće definirati vrml, iv i wrl datoteke i uvesti modele.

Environment (okruženje) – sadrži više objekata i robota. Može definirati i neka GUI svojstva kao što su početna lokacija kamere i boja pozadine. Okruženje omogućuje stvaranje OpenRAVE sučelja.

KinBody – temeljni cilj je da svi ostali objekti proizlaze iz njega. Kinematičko tijelo se sastoji od zbirke krutih tijela i zglobova koji povezuju ta tijela.

Robot – osnovni robot proizlazi iz **KinBody**. Obično svaki robot ima barem jednu KinBody deklaraciju unutar njega. Robot također može sadržavati **Manipulator** i **AttachedSensor** koji opisuju manipulacije i senzore robota.

3.4.1. Okruženje

Okruženje (environment) služi za definiranje radnog okruženja. U njemu se pozicioniraju svi roboti, alati i predmeti koji okružuju robote. Također se definiraju i pozicije kamera za pogled. Ovdje su dani neki bitni parametri koji se koriste u definiranju okruženja.

environment - svojstva: file

- bkgndcolor - 3 float - pozadinska boja okruženja, RGB
- camrotaxis - 4 float - početne rotacije kamere oko osi (Xr, Yr, Zy, deg)
- camtrans - 3 float - početna pozicija kamere (X, Y, Z)
- kinbody - svojstva: name, file – učitavanje predmeta u xml formatu
- robot - svojstva: name, file - učitavanje datoteke robota u xml formatu
- planner - svojstva: type, file
- sensorsystem - svojstva: type, file
- controller - svojstva: type, file, robot, args
- module - svojstva: type, file
- inversekinematicssolver - svojstva: type, file
- physicsengine - svojstva: type, file
- sensor - svojstva: type, file

- collisionchecker - svojstva: type, file
- trajectory - svojstva: type, file
- viewer - svojstva: type, file
- server - svojstva: type, file

Primjer dijela neurokirurškog okruženja definiranog u OpenRAVE-u.

```
<Environment>
	<bkgndcolor>0.3 0.7 0.8</bkgndcolor>
	<camtrans>-1.5 -1.5 0.5</camtrans>
	<camrotaxis>0.8 -0.4 -0.5 90</camrotaxis>
	<Robot file="robots/fsb/mate.robot.xml" name="Fanuc">
		<translation>0 0.3 0</translation>
	</Robot>
	<Robot file="robots/fsb/kuka.robot.xml" name="Kuka_LBR_4+">
		<translation>0 -0.5 0</translation>
	</Robot>
	<KinBody name="markerska_plocica">
		<Body type="ststic">
			<Geom type="trimesh">
				<Data>robots/fsb/pribor/plocica.wrl 1.0</Data>
				<Render>robots/fsb/pribor/plocica.wrl 1.0</Render>
			</Geom>
			<Translation>0.51 -0.3 0.19</Translation>
			<RotationAxis>0 0 1 180</RotationAxis>
		</Body>
	</KinBody>
</Environment>
```

3.4.2. KinBody

Svi objekti koji se žele učitati u OpenRAVE moraju biti definirati kao xml format. Svaki objekt se definira kao KinBody. Roboti se također najprije definiraju na ovaj način.

kinbody - atributi: name, file, prefix, type

- body - svojstva: name, type (dynamic, static) - dinamička geometrija je podložna fizikalnim zakonima, dok se statička geometrija neće premjestiti djelovanjem fizikalnih sila
    - geom - svojstva: name, type (box, sphere, trimesh, cylinder)
    - ambientcolor - 3 float - boja geometrije
    - diffusecolor - 3 float - boja geometrije
    - extents - 3 float [box] - dimenzije kutije, pola širine, visine dužine
    - height - float [cylinder] - visina cilindra definirana u smjeru y osi
    - radius - float [cylinder, sphere]
    - rotationaxis - 4 float - rotacija geometrije (Xr, Yr, Zr, deg)
    - rotationmat - 9 float - 3x3 matrica rotacije geometrije
    - translation - 3 float - translacija tijela
    - transparency - 1 float - transparentnost geometrije, 0 je neprozirno
  - mass - svojstva: type (box, sphere, custom, mimicgeom)
    - com - 3 float – centar težišta tijela
    - density – float
    - inertia - 9 float – 3x3 matrica inercije
    - total – float – masa tijela u kg
- joint - svojstva: type (hinge, slider, universal, hinge2, spherical), circular(true, false), mimic_pos (matematička formula), mimic_vel (matematička formula), mimic_accel (matematička formula)
  - axis - 3 float [hinge, slider] – osi rotacije ili klizanja zgloba, (0,0,1) znači rotaciju oko Z osi
  - body – string – služi za označavanje dijelova između kojih se definira zglob
  - limits - 2*dof floats – ograničenje kliznog zgloba u metrima

  - limitsdeg - 2*dof floats – ograničenje zgloba u stupnjevima
  - limitsrad - 2*dof floats – ograničenje zgloba u radijanima
  - maxvel - dof floats – maksimalna brzina zgloba u m/s
  - maxveldeg - dof floats – maksimalna brzina zgloba u °/s
  - maxaccel - dof floats – maksimalno ubrzanje zgloba u m/s^2
  - maxacceldeg - dof floats – maksimalno ubrzanje zgloba u $°/s^2$
  - offsetfrom – ime tijela/linka – u odnosu na koji dio se odnosi translacija
  - resolution – float – rezolucija zgloba u m ili rad
  - maxtorque - dof floats – maksimalni moment zgloba u Nm
- kinbody - svojstva: name, file, prefix – učitavanje objekta
- mass - svojstva: type (box, sphere, custom)
  - density - float
  - radius - float
  - total – float – masa cijelog KinBody-a u kg
- rotationaxis - 4 float
- rotationmat - 9 float
- translation - 3 float
- transparency - 1 float

Primjer definiranja tijela objekta za OpenRAVE.

```
<KinBody name="markerska_plocica">
    <Body type="static">
        <Geom type="trimesh">
            <Data>robots/fsb/pribor/plocica.wrl 1.0</Data>
            <Render>robots/fsb/pribor/plocica.wrl 1.0</Render>
        </Geom>
    </Body>
</KinBody>
```

3.4.3. Robot

Da bi se robot mogao kretati potrebno ga je definirati. Da bi se mogao definirati najprije treba definirati tijelo robota kako je prikazano poglavlje ispred.

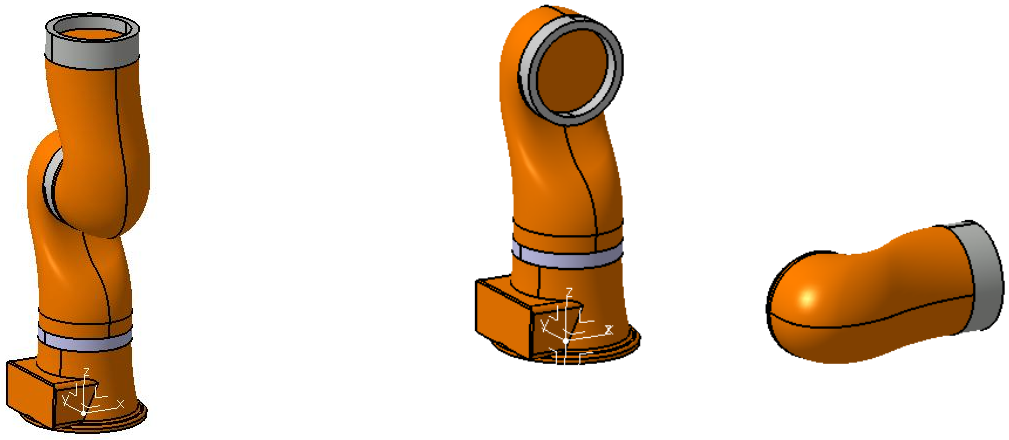
robot - svojstva: name, file, prefix, type

- attachedsensor - svojstva: name – služi za dodavanje senzora, npr. kamera, laser
    - link – dio na koji se veže senzor
    - rotationaxis
    - rotationmat
    - translation
    - sensor - svojstva: type – vrsta senzora
- kinbody – name – učitavanje xml tijela robota i tijela
- manipulator - svojstva: name – definira se kretanje robota
    - base – ime linka – ime dijela koji je baza robota
    - effector – ime linka – ime linka koji je zadnji u lancu
    - gripperjoints - k imena zglobova
    - closingdirection - k floats - – pozicije prihvatnice kada je zatvorena
    - direction - 3 floats – smjer prilaženja prihvatnice objektu
    - rotationaxis - 4 floats
    - rotationmat - 9 floats
    - translation - 3 floats
- robot - name, file, type – učitavanje definiranog robota
- rotationaxis - 4 float
- rotationmat - 9 float
- translation - 3 float

Primjer definiranja robota prikazan je u sljedećem poglavlju.

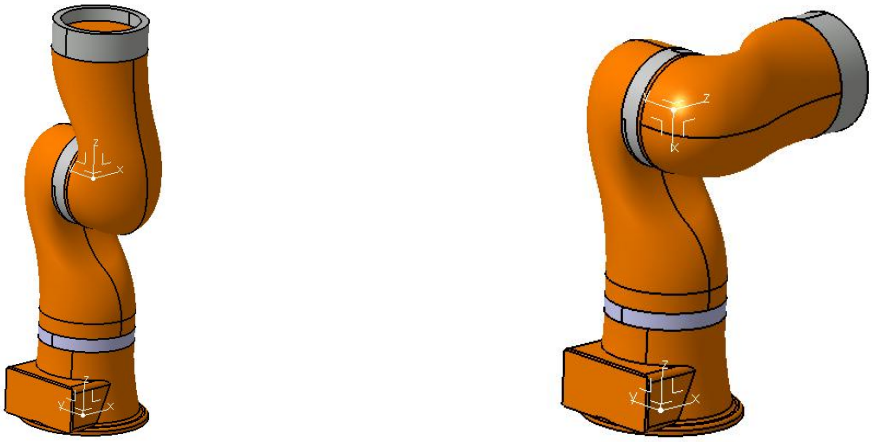
3.5. Konfiguriranje manipulatora

Manipulator koji se želi koristiti u OpenRAVE potrebno je konfigurirati u xml formatu. OpenRAVE prihvaća wrl i iv formate CAD modela. Kod konstrukcije CAD modela manipulatora potrebno je koordinatni sustav svakog dijela postaviti u točku oko koje se pojedini zglob giba. To je stoga što OpenRAVE nudi mogućnost pokretanja zglobova samo oko koordinatnih osi zglobova.



Slika 28. Primjer krivo postavljenih koordinatnih sustava

Ovdje je prikazan primjer kako OpenRAVE konfigurira zglobove. Na prvoj slici [Slika 28] je prikazano gibanje kada koordinatni sustavi nisu na točki oko koje se dio giba, nego su oba koordinatna sustava preklopljena, dok druga slika [Slika 29] prikazuje isto gibanje kada su koordinatni sustavi ispravno postavljeni.



Slika 29. Primjer kada su koordinatni sustavi dobro postavljeni

Kako je već spomenuto, konfiguracija manipulatora se sprema u xml formatu koji se kasnije može učitavati u OpenRAVE. U njemu su sadržani svi dijelovi manipulatora te su definirani zglobovi.

Slijedi primjer konfiguriranja manipulatora, Fanuc LR Mate 200iC 5L. Najprije se definira tijelo manipulatora i veze između njih.

```
<KinBody name="Fanuc LR Mate 200iC 5L">>
        <Body name="base" type="dynamic"> ... </Body>
        <Body name="axis1" type="dynamic"> ... </Body>
        <Joint name="J1" type="hinge"> ... </Joint>
        <Body name="axis2" type="dynamic"> ... </Body>
        <Joint name="J2" type="hinge"> ... </Joint>
        ...
</KinBody>
```

Prikazana je struktura zapisa manipulatora. Svaki manipulator mora biti sastavljen od tijela manipulatora (KinBody) i upravljanja (Manipulator). KinBody se sastoji od svih dijelova manipulatora (Body) i zglobova između njih (Joint).

```
<Body name="base" type="dynamic">
        <Geom type="trimesh">
                <Data>robots/200iC/200iC_base.wrl 1.0</Data>
                <Render>robots/200iC/200iC_base.wrl 1.0</Render>
        </Geom>
        <Translation>0 0 0</Translation>
        <RotationAxis> 0 0 0 90</RotationAxis>
        <mass type="custom">
                <total>39.689</total>
                <com>-0.024 0 0.077</com>
        </mass>
</Body>
```

U Body-u je definiran dio manipulatora. Svako tijelo mora sadržavati ime (name) i tip (type). Tip tijela može biti dynamic i static. Pošto je ovdje riječ o manipulatoru, tip mora biti dynamic.

Unutar Geom zagrada stavljaju se Data i Render. U njima se učitava model tijela manipulatora. U putanji ne smiju biti prazna mjesta, stoga je najbolje CAD modele robota spremiti u OpenRAVE mapu robots ili u korijen diska. Također i u nazivu dijela ne smiju biti razmaci. Broj iza putanje predstavlja vrijednost uvećanja ili smanjenja dijela. Vrijednost 1 znači da će model u OpenRAVE-u biti jednak CAD modelu. Npr. 1,5 bi značilo da je model u OpenRAVE-u 50% veći u odnosu na učitani model.

Translation predstavlja translatiranje tijela. Varijable koje se upisuju predstavljaju translaciju u smjeru x, y i z osi u metrima. Kod učitavanja dijelova, koordinatni sustavi svih dijelova se postave u koordinatni sustav okruženja. Kako svi dijelovi moraju imati koordinatni sustav u točci u kojoj se gibaju, tako su svi dijelovi u početku na istom mjestu. Translacijom je potrebno tijelo translatirati točno na njegovu poziciju u sklopu.

Ako osi nekog tijela nisu isto orijentirane kao i glavni koordinatni sustav, te zbog toga tijelo u početku nije pravilno orijentirano mora se rotirati. Za to se koristi RotationAxis. Ova naredba sadrži četiri varijable. Prve tri predstavljaju skalarne vrijednosti za rotacije oko x, y i z osi, dok četvrta predstavlja kut rotacije u stupnjevima. Npr. ako upišemo '0 1 0 90' rotira se tijelo oko osi y za 90°. Ako je potrebno tijelo rotirati oko više osi za različite kutove može se staviti više naredbi te u svakoj rotirati tijelo oko jedne osi.

U varijabli mass definiraju se podaci o masi tijela. Pod type u slučaju manipulatora stavlja se custom. Unutar ove varijable definiramo total koji predstavlja masu tijela u kilogramima. Unutar com se definira težište tijela. Također može biti definirana i inercija. Ona se definira unutar inertion i definira se 3x3 matricom.

Time se definiraju osnovne značajke tijela. Nakon što se učita još jedan dio moguće je definirati zglob između njih.

```
<Joint name="J1" type="hinge">
        <Body>base</Body>
        <Body>axis1</Body>
        <offsetfrom>axis1</offsetfrom>
        <axis>0 0 1</axis>
        <limitsdeg>-170 170</limitsdeg>
        <maxveldeg>5</maxveldeg>
</Joint>
```

Prvo je potrebno definirati ime i tip zgloba. Kod manipulatora kao tip zgloba najčešće koristimo hinge za kružno gibanje zgloba i slider za klizno translacijsko gibanje zgloba. U Body se definiraju dijelovi koji su povezani zglobom. U offsetfrom se definira koji se dio giba u odnosu na drugi. S axis se definira os oko koje se dio naveden u offsetfrom giba. Tri varijable određuju osi x, y i z. Iz tog je razloga potrebno koordinatni sustav svakog dijela staviti u točku gibanja dijela. Limitsdeg određuje ograničenja zglobova u stupnjevima. Kod translatornog gibanja umjesto limitsdeg koristimo limits gdje definiramo granice u metrima. Maxveldeg određuje maksimalnu brzinu gibanja zglobova izraženu u stupnjevima.

U Joint moguće je definirati još nekoliko vrijednosti. Naredbom resolution se definira rezolucija zgloba. Ta vrijednost je u onoj jedinici u kojoj se definiraju ograničenja.

Na kraju se dodaju parovi robota koji imaju mogućnost doći u kontakt jedni s drugima. To se dodaje zbog bržeg računanja inverznog kinematičkog problema.

```
        <adjacent>base axis4</adjacent>
        <adjacent>base axis5</adjacent>
        <adjacent>base axis6</adjacent>
        <adjacent>axis1 axis2</adjacent>
        <adjacent>axis2 axis3</adjacent>
        <adjacent>axis2 axis4</adjacent>
        <adjacent>axis4 axis5</adjacent>
```

Nakon što je kreirano tijelo manipulatora kreira se i robot.

```
<Robot name="Fanuc_LR_Mate_200iC_5L">
        <KinBody file="robots/fsb/Fanuc_LR_Mate_200iC_5L.xml"/>
        <Manipulator name="arm">
                <base>base</base>
                <effector>axis6</effector>
                <direction>0 0 0</direction>
        </Manipulator>
</Robot>
```

Ovo je prikaz strukture robota. Najprije se učita tijelo robota u xml formatu. Zatim se definira manipulator. Potrebno je definirati bazu manipulatora i zadnji član u vezi. Direction označava smjer prilaza robota objektu.

3.6. Konfiguriranje prihvatnice

Kinbody prihvatnice konfiguriramo na jednak način kako je objašnjeno za robot. Na isti način se učitaju dijelovi i konfiguriraju zglobovi između njih. Kako prihvatnice često imaju linearno gibanje stoga umjesto parametra *limitdeg* koristi *limit*. Također, pošto se svi prsti prihvatnice gibaju zajedno, moguće je konfigurirati zglobove tako da su povezani s nekim zglobom kojeg upravljamo. To je prikazano u idućem primjeru.

```
<Joint name="JF2" type="slider" enable="false" mimic_pos="JF1" mimic_vel="|JF1 1">
        ...
</Joint>
```

Kao tip zgloba koristi se klizač (slider). Parametar *mimic_pos* matematički opisuje zavisnost zgloba dva o zglobu pod nazivom JF1. Parametar *mimic_vel* označava brzinu zgloba.

Kad se želi povezati prihvatnica s robotom to se obavlja na sljedeći način.

```
<Robot name="robot3">
        <KinBody file="robots/fsb/Fanuc_LR_Mate_200iC_5L.xml"/>
        <KinBody file="robots/fsb/Schunk_MPG32.xml"/>
        <Kinbody>
                <body name="handbase">
                        <offsetfrom>axis6</offsetfrom>
                        <Translation>0 0 -0.007</Translation>
                        <RotationAxis> 0 0 1 90</RotationAxis>
                </body>
                <joint name="dummyhand" type="hinge" enable="false">
                        <body>axis6</body>
                        <body>handbase</body>
                        <limits>0 0</limits>
                </joint>
        </Kinbody>
        <Manipulator name="fanuc3">
                <base>base</base>
```

```
            <effector>axis6</effector>
            <Translation>0 0 0</Translation>
            <joints>JF1</joints>
            <closingdirection>0.008</closingdirection>
            <direction>0 0 0</direction>
        </Manipulator>
</Robot>
```

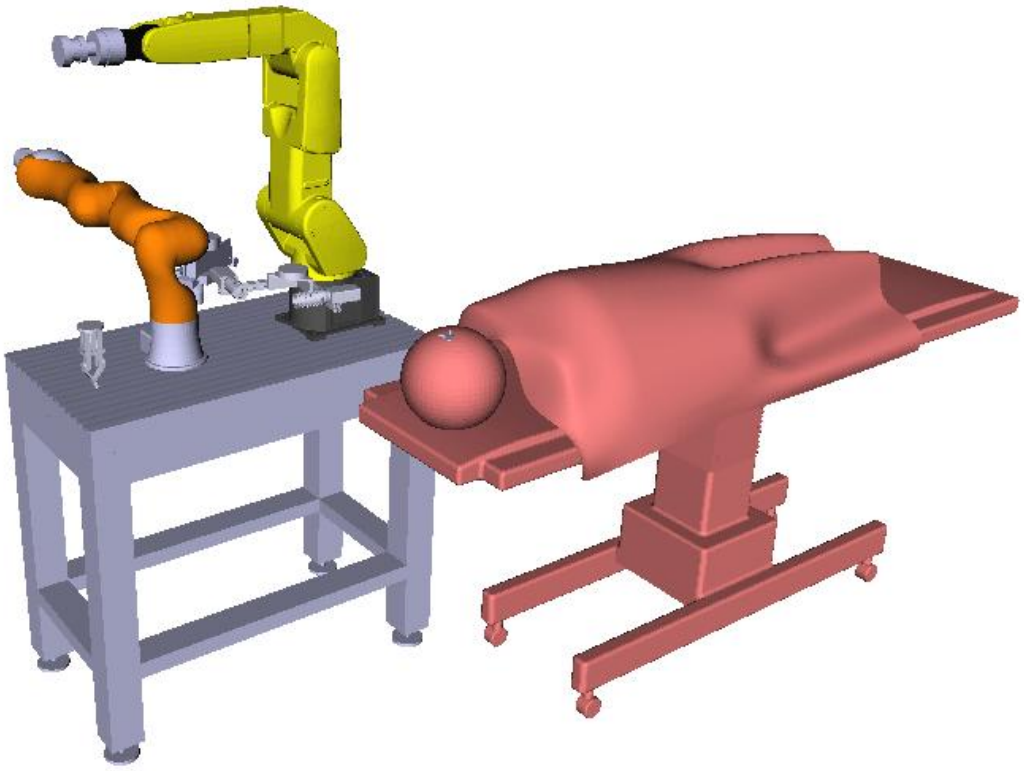
Slika 30. Prihvatnica kreirana u OpenRAVE-u

Najprije se učitaju robot i prihvatnica, te se zatim kreira veza između njih. To se mora učiniti tako da između prirubnice robota i baze prihvatnice kreira zglob kojem se postave granice na nulu. U dijelu *Manipulator* se dodaje zglob prihvatnice koji se upravlja, te je potrebno definirati vrijednost pozicije koju prihvatnica postiže kada se naredbom zadaje zatvaranje prihvatnice.

3.7. Stvaranje okruženja za rad

Okruženje se u OpenRAVE-u također definira u xml formatu. U njega se dodaju roboti, objekti i svo okruženje. Također se definira početna pozicija kamere i boja okoline. Unutar okruženja se definiraju sve pozicije objekata. Ono se učitava kod pokretanja programa.

```
<Environment>
    <bkgndcolor>1 1 1</bkgndcolor>
    <camtrans>-1.5 -1.5 0.5</camtrans>
    <camrotaxis>0.8 -0.4 -0.5 90</camrotaxis>
    <Robot file="robots/fsb/robot3.robot.xml"
        name="Fanuc_LR_Mate_200iC_5L">
        <translation>0 0 -0.330</translation>
        <RotationAxis>0 0 0 90</RotationAxis>
    </Robot>
</Environment>
```



Slika 31. OpenRAVE laboratorijsko okruženje

4. UPRAVLJAČKE MOGUČNOSTI OPENRAVE-A

Manipulatorima je moguće upravljati na nekoliko načina. Prvi način je da se upravlja unutarnjim koordinatama robota, tj. da se upravlja kutom zakreta svakog pojedinog zgloba robota. To je jednostavno upravljanje, ali je dovođenje u željenu poziciju teže.

Drugi način je pomoću vanjskih koordinata. Zadaje s se pozicija vrha alata ili prihvatnice u prostoru, te orijentacija alata oko tri osi. Iz tih podataka najprije se treba izračunati matrica homogenih transformacija iz koje robotski sustav poznavajući kinematiku robota izračunava unutarnje koordinate robota, tako da vrh alata postavi u traženu točku.

Upravljanje pomoću unutarnjih koordinata omogućeno je na dva načina. Prvi način je upravljanje točno određenim zglobovima. Prvom naredbom definira se robot. Drugom naredbom pokreče se robot i to tako da prvi vektor sadrži kutove zakreta zglobova u radijanima, dok drugi vektor definira na koje se zglobove odnose vrijednosti u prvom vektoru. Valja napomenuti kako se ovom naredbom robot giba maksimalnom brzinom i nije nju moguće kontrolirati.

```
robot = env.GetRobots()[0]
robot.SetDOFValues([J1, J2, J3, J4],[0,1,2,3])
```

Drugi način upravljanja unutarnjim koordinatama je upravljanje svim zglobovima. Drugom se naredbom definira ruka robota kao bazni manipulator. Trećom se naredbom pokreće manipulator za kutove zakreta zglobova definiranih vektorom goal. Ovdje je potrebno definirati kutove zakreta za sve zglobove manipulatora. Kad se koristi ova naredba, robot se giba brzinom koja se definira.

```
robot = env.GetRobots()[0]
manip=interfaces.BaseManipulation(robot)
manip.MoveManipulator(goal=[J1,J2,J3,...])
```

Upravljanje vanjskim koordinatama se često primjenjuje u robotici. Kod ovog načina definiraju se koordinate točke u koju želimo dovesti prihvatnicu te tri kuta orijentacije. Ali da se dade naredba robotu potrebno je izračunati matricu homogenih transformacija, koja sadrži podatke o poziciji i orijentaciji. Ona se dobiva na sljedeći način.

```
matrixFromAxisAngle([Xr,Yr,Zr])
```

Kod korištenja ove naredbe primjećeno je da ne radi ispravno. Kad se uvrsti rotacija oko više osi dobiveni rezultat nije ispravan. Stoga je potrebno izračunati matricu rotacija za svaku osi, te zatim pomnožiti sve tri matrice kako je prikazano u idućem primjeru.

```
rx=matrixFromAxisAngle([Xr,0,0])
ry=matrixFromAxisAngle([0,Yr,0])
rz=matrixFromAxisAngle([0,0,Zr])
numpy.array(rz*ry*rx)
```

Nakon toga se u četvrti stupac dodaju vrijednosti pozicije i time smo dobili matricu homogenih transformacija.

Kod upravljanja vanjskim koordinatama najprije je potrebno generirati inverzni kinematički model. Njegovim rješavanjem, na temelju poznate geometrije robota izračunavaju se unutarnje koordinate robota. To se radi sljedećim blokom naredbi. Prva naredba učitava prvog robota i imenuje ga s robot. Druga naredba generira inverzni kinematički model za robot. Sljedeće dvije naredbe služe ako konfiguracija robota nije u bazi, te se potom generira model.

```
robot = env.GetRobots()[0]
ikmodel=databases.inversekinematics.InverseKinematicsModel(robot,iktype=IkPar
ameterization.Type.Transform6D)
if not ikmodel.load():
        ikmodel.autogenerate()
```

Nakon što je generiran inverzni kinematički model moguće je upravljanje robotom. Prva naredba aktivira manipulator robota. Druga naredba predstavlja matricu homogenih transformacija. Iduća naredba izračunava kutove zakreta pojedinih zglobova temeljem kinematike robota. Posljednja naredba pokreče robota i vodi ga na zadanu poziciju.

```
manip = robot.GetActiveManipulator()
mht = numpy.array([[ix,jx,kx,px],[ iy,jy,ky,py],[ iz,jz,kz,pz],[0,0,0,1]])
sol = manip.FindIKSolution(mht, IkFilterOptions.CheckEnvCollisions)
robot.SetDOFValues(sol,manip.GetArmIndices())
```

Drugi način upravljanja je sljedeći. Struktura i značenje naredba je jednako kao i prethodni primjer. Osim korištenja drugih naredbi.

```
manip = interfaces.BaseManipulation(robot)
```

```
mht = numpy.array([[ix,jx,kx,px],[ iy,jy,ky,py],[ iz,jz,kz,pz],[0,0,0,1]])
res = manip.MoveToHandPosition(matrices=[mht],execute=False,seedik=10)
robot.GetController().SetPath(res)
```

U svim ovim primjerima robot se između dvije točke kreće putanjom koju sam odredi prema nekim kriterijima (npr. najbrži dolazak u drugu točku, najmanje potrošene energije,...). U robotici se često želi da se robot između dvije točke kreće linearno. To se postiže korištenjem sljedećih primjera.

Linearno kretanje u određenom smjeru za određenu udaljenost. Vektorom smjer određuje se smjer gibanja robota. Taj vektor određuje težinu osi u kojoj se robot giba. Npr. kako je u primjeru, robot bi se gibao u smjeru z-osi za 200 mm.

```
smjer = array((0,0,1))
ruka.MoveHandStraight(direction=smjer,stepsize=0.01,minsteps=1,maxsteps=20)
```

Kad se želi robota pomaknuti za 300 mm po y osi i 200 mm po z osi. Smjer se definira udaljenostima u smjerovima osi. Varijabla d je duljina puta koji robot treba prevaliti. Stepsize je veličina koraka a maxsteps je broj koraka koji treba učiniti da se dođe u konačnu točku.

```
smjer = array((0,0.3,0.2))
d=math.sqrt(smjer[0]**2+smjer[1]**2+smjer[2]**2)
stepsize=0.00001
maxsteps=d/stepsize
ruka.MoveHandStraight(direction=smjer,stepsize=stepsize,minsteps=1,maxsteps=m
axsteps)
```

Ukoliko se želi sljedeća točka zadati koordinatama umjesto smjera i udaljenosti to se može učiniti na sljedeći način. Varijabla T je trenutna pozicija robota. T1 je vektor koordinata točke u koju se robot želi poslati. Smjer se izračunava iz razlike ta dva vektora, tako da se dobiju udaljenosti točaka po osima. Zatim se izračuna udaljenost tih točaka i broj koraka. Naredbom drawlinelist se iscrtava putanja kojom će robot proći. Prvi parametar su početna i krajnja točka, zatim debljina linije i na kraju vektor za boju linije.

```
T=robot.GetLinks()[6].GetTransform()
T1=numpy.array([0.485,-0.100,0.345])
smjer=T1-T[0:3,3]
```

```
d=math.sqrt(smjer[0]**2+smjer[1]**2+smjer[2]**2)
stepsize=0.00001
maxsteps=d/stepsize
env.drawlinelist(array([T[0:3,3],T1]),2,[0,1,0])
ruka.MoveHandStraight(direction=smjer,stepsize=stepsize,minsteps=1,maxsteps=m
axsteps)
```

Pošto se svugdje za kutove koriste radijani, a praktičnije je zadavanje u stupnjevima mogu se iskoristiti sljedeće naredbe koje proračunava stupnjeve u radijane.

```
math.radians()
numpy.radians()
```

Kod upravljanja robota unutarnjim koordinatama žele se znati vanjske koordinate u kojima se robot nalazi. Matrica homogenih transformacija pojedinog zgloba robota dobiva se sljedećim naredbama.

```
robot.GetLinks()[6].GetTransform()
manip.GetEndEffectorTransform()
```

U prvoj naredbi broj koji se upisuje određuje link robota za koji se želi matrica homogenih transformacija. Kod druge naredbe je manipulator robota definiran te se vraća matrica homogenih transformacija posljednjeg linka.

Važno je napomenuti kako je potrebno koordinatni sustav krajnjeg člana robota ili alata koji se na njemu nalazi postaviti na mjesto gdje se stvarni koordinatni sustav i nalazi. Ako to nije slučaj, dobivene koordinate nisu ispravne, jer one pokazuju gdje se nalazi koordinatni sustav tog člana.

Kod upravljanja vanjskim koordinatama često se žele znati unutarnje koordinate robota kako bi se moglo na primjer upravljati pravim robotom. U tom slučaju možemo koristiti:

```
robot.GetDOFValues()
```

Također i varijabla sol u prvom primjeru upravljanja robotom vanjskim koordinata sadrži kutove zakreta zglobova robota u radijanima.

Za preračunavanje kutova iz radijana u stupnjeve mogu se koristiti sljedeće naredbe.

```
math.degrees()
numpy.degrees()
```

Kod upravljanja robotom poželjno je regulirati i brzinu gibanja robota. Maksimalna brzina se definira u konfiguraciji manipulatora, ali se može postaviti i korištenjem naredbe u konzoli. Prva naredba vraća vektor s maksimalnim brzinama zglobova a drugom naredbom se postavljaju brzine robota. Vektor mora sadržavati onoliko varijabli koliko ima stupnjeva slobode gibanja.

```
robot.GetDOFMaxVel()
robot.SetDOFVelocityLimits([J1,J2,...])
```

OpenRAVE omogućuje upravljanje radom prihvatnice. Za razliku od robota koji se upravljaju konačnom pozicijom, prihvatnica se upravlja naredbama za zatvaranje i otvaranje. To omogućuje zatvaranje prihvatnice dok ne dođe do kolizije s predmetom kojeg se želi uhvatiti.

```
robot=env.GetRobots()[0]
taskprob = interfaces.TaskManipulation(robot)
taskprob.CloseFingers()
taskprob.ReleaseFingers()
```

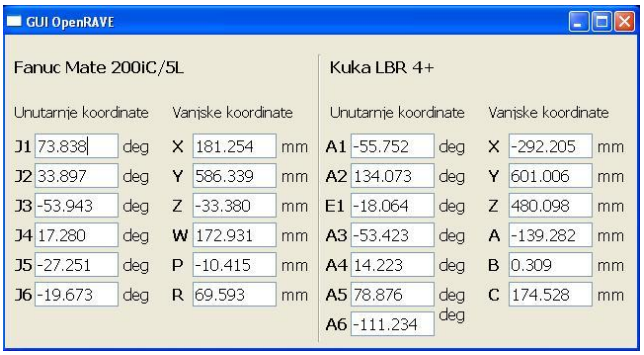
5. PRIMJENA OPENRAVE-A U NEUROKIRURŠKOM OKRUŽENJU

Zadatak ovog rada je implementacija OpenRAVE-a na neurokirurški sustav u laboratoriju. U OpenRAVE-u su konstruirana dva robota, Fanuc LR Mate 200iC/5L i Kuka LWR 4+ te alati koje roboti koriste i operacijsko okruženje. Računalo na kojem je OpenRAVE, povezan je s robotima pomoću LAN mreže. Na svaki zahtjev programa, roboti šalju unutarnje koordinate, tj. kutove zakreta svih zglobova i oznaku alata koji je priključen na njima. Ti podaci se obrađuju na računalu te se zatim u OpenRAVE-u simulira njihova pozicija. Također je izrađen jednostavan GUI [Slika 32] u kojem se pokazuju trenutne pozicije robota u vanjskim i unutarnjim koordinatama.

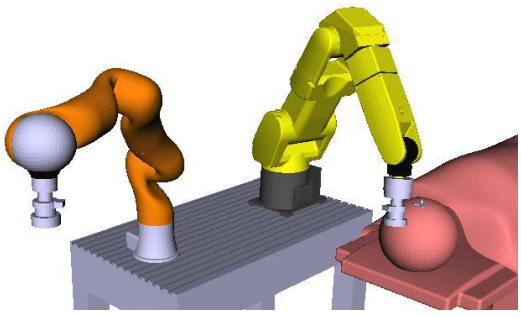
Za implementaciju programa najprije je potrebno konstruirati i konfigurirati oba robota i alate kojima se služe.

Kako preuzeti modeli nisu imali pravilno postavljene koordinatne sustave, bilo je potrebno neke dijelove robota ponovno konstruirati. To je bilo potrebno zbog toga što OpenRAVE kod konfiguriranja zglobova ne može dva dijela vezati pomoću geometrijskih obilježja, kako se radi npr. u CATIA programskom CAD paketu, nego je moguće jedino dva dijela vezati koordinatnim sustavima. Svaki dio treba imati koordinatni sustav u onoj točki oko koje se kreće u odnosu na drugi dio.

Izrada manipulatora robota je opisana u poglavlju 3.5 dok je kod za oba robota priložen u dodacima.



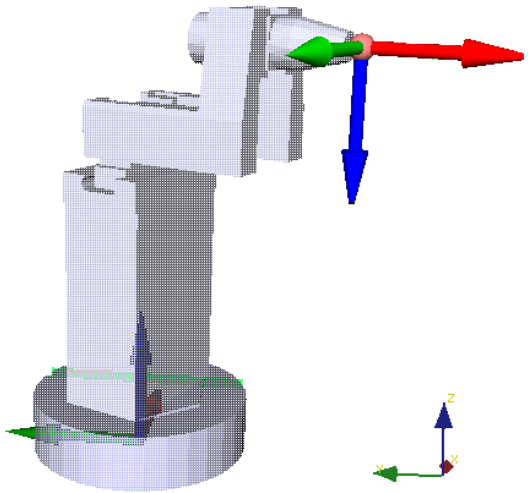
Slika 32. GUI za prikaz koordinata robota



Slika 33. OpenRAVE prikaz neurokirurškog okruženja

Kod konstruiranja alata najbolje je koordinatni sustav postaviti na mjesto gdje se nalazi i stvarni koordinatni sustav, kako bi se olakšalo dobivanje stvarne pozicije alata. Ako su poznate sve transformacije alata a koordinatni sustav nije u vrhu alata te ne možemo dobiti

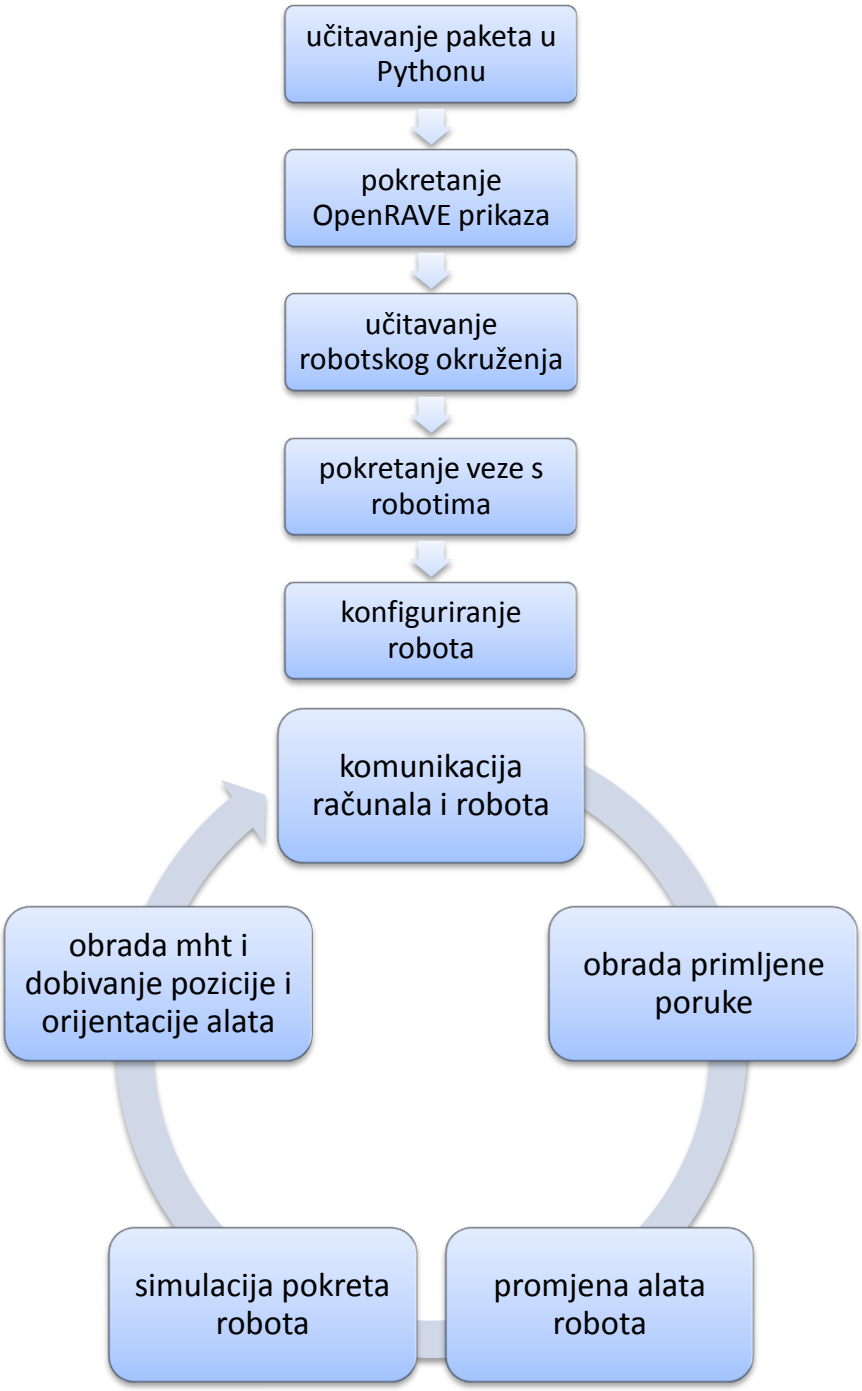
stvarnu poziciju, potrebno je matematički izračunati poziciju vrha alata. Ali kada se koordinatni sustav nalazi na vrhu alata javlja se problem pozicioniranja alata na robot, jer sve transformacije koje izvodimo naknadno s alatom odnose se na koordinatni sustav okruženja, te ako rotiramo predmet oko neke osi on se rotira oko koordinatnog sustava okruženja. Iz tog razloga alat je konstruiran iz dva dijela [Slika 34]. Prirubnica je jedan dio te je koordinatni sustav postavljen tako da se postavi na robota bez potrebe za rotacijama. Drugi dio je ostatak alata kojemu je koordinatni sustav postavljen u vrh alata. Nakon toga se oba dijela povežu kao robot, ali se ograničenja na zglob između njih postave na 0, čime je onemogučeno kretanje. Time se dobije prihvatnica s dva koordinatna sustava. Prvi koordinatni sustav koji se nalazi na prirubnici hvataljke služi za povezivanje s robotom, dok drugi koordinatni sustav označava stvarni koordinatni sustav hvataljke koji omogućuje dobivanje točnih vanjskih koordinata.



Slika 34. Mjerna kugla iz dva dijela

Također su konstruirane prirubnice za oba robota [Slika 36 i Slika 37]. Prirubnica za Kuku nema senzor sila iz razloga što su senzori već ugrađeni na svaku os robota.

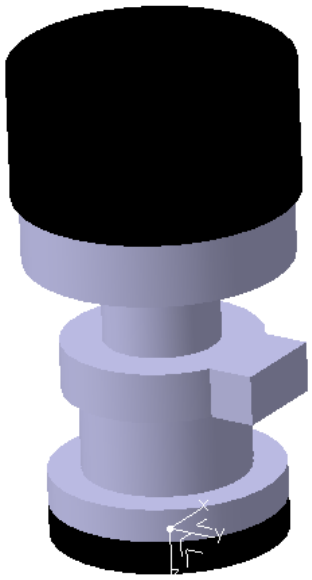
Stablom [Slika 35] je prikazana struktura programa. Iz njega je vidljiv tijek odvijanja programa. Nakon pokretanja Python skripte potrebno je oko 10 sekundi za pokretanje i učitavanje svih potrebnih paketa u Pythonu, te uspostavljanje veze s robotima. Nakon toga program ulazi u petlju gdje komunicira s robotima i izvršava kretanje robota u simulaciji.



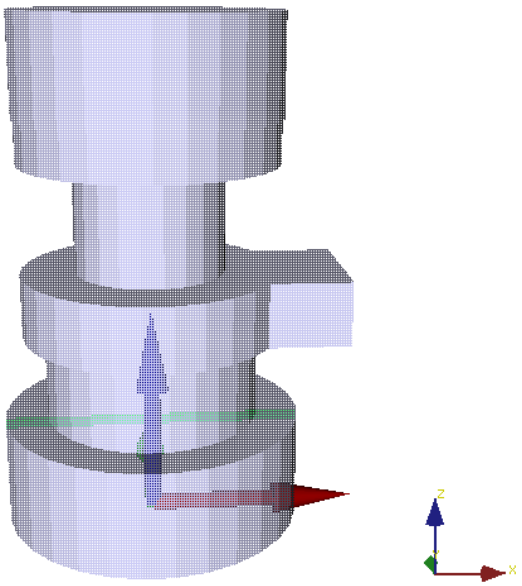
Slika 35. Stablo programa

Prvi korak nakon učitavanja svih potrebnih paketa vezanih za rad OpenRAVE-a je kreiranje veze između robota i računala. U ovom slučaju roboti su serveri dok je računalo klijent. Komunikacija se odvija tako da OpenRAVE pošalje poruku zahtijeva za poziciju robotima, a oni odgovaraju s vlastitim unutarnjim koordinatama. U svrhu komunikacije, na upravljačkom računalu Kuke izrađen je program koji na svaki zahtjev šalje koordinate robota. Na Fanucu je također isprogramiran program koji šalje koordinate.

```
ADDRfanuc = ('192.168.123.25',1010)        #IP adresa i PORT
clifanuc = socket.socket(socket.AF_INET,socket.SOCK_STREAM)
clifanuc.connect((ADDRfanuc))       #uspostavljanje veze
clifanuc.sendall('zahtijev')     #slanje poruke
clifanuc.recv(4096)    #primanje poruke
```



Slika 36. Prirubnica Fanuca

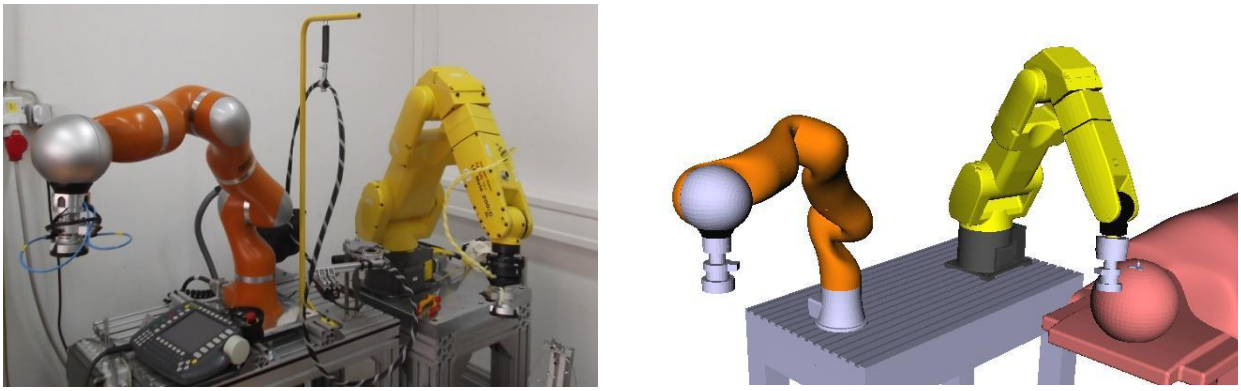


Slika 37. Prirubnica Kuke u OpenRAVE-u

Nakon primanja poruke u kojoj su koordinate robota potrebno je iz poruke odvojiti koordinate i broj alata koji je priključen na robotu. Kako robot šalje koordinate u stupnjevima, a OpenRAVE radi u radijanima, potrebno je pretvoriti koordinate u radijane i narediti robotu da se pomakne na zadanu poziciju. Prije toga se provjerava alat koji se nalazi na robotu. Kako je u stvarnom sustavu moguće da se alati pomaknu te nisu svaki put na istom mjestu i isto orijentirani, alat se u OpenRAVE-u ne odlaže, nego se samo dodaje i miče s robota.

Nakon što se robot pomakne na novu poziciju, izračunavaju se vanjske koordinate robota. Koordinate se izračunavaju tako da se pomoću funkcije iz OpenRAVE-a dobije matrica homogenih transformacija koordinatnog sustava vrha alata. Iz tog je razloga bilo potrebno konstruirati alat iz dva dijela. Nakon toga se matrica obrađuje, te se dobiva pozicija i orijentacija koordinatnog sustava alata.

Vanjske koordinate robota zapisuju se u txt datoteku na disku računala gdje ih preuzima GUI aplikacija koja ih prikazuje u prozoru. GUI aplikacija je odvojena od OpenRAVE koda iz razloga što se GUI aplikacija ne može osvježiti u svakom koraku dok je radila u istoj skripti.



Slika 38. Usporedni prikaz robota u laboratoriju i u OpenRAVE-u

GUI aplikacija [Slika 32] je kreirana u QT Designer-u [Slika 27]. Da bi se aplikacija mogla pokrenuti u Pythonu potrebno je instalirati modul PyQt4. Unutar ovog modula postoji kompailer koji pretvara .ui datoteku kreiranu u QT Designer-u u Python kod. On se pokreće tako da u Comand Promt-u odemo u mapu u kojoj se nalazi datoteka koju želimo kompajlirati. Zatim se unese naredba *pyuic4 GUI.ui > gui.py*. Prva varijabla (GUI.ui) je datoteka kreirana u QT Designer-u, a druga varijabla je naziv nove datoteke. Dobivenu Python datoteku potrebno je nakon toga urediti jer kompajler generira samo prozor i njegove elemente, a potrebno je dodati naredbe tipične za Python.

Aplikacija je veoma jednostavnog izgleda i pruža pregledniji prikaz unutarnjih i vanjskih koordinata za oba robota. Kada se promjeni alat na robotu tada se prikazuju vanjske koordinate tog alata. Za izvršavanje jednog ciklusa aplikacije potrebno je svega 4 ms, što je prebrzo jer blokira pisanje podataka u txt datoteku stoga je dodano zadržavanje od 50 ms. Aplikacija i dalje radi prilično brzo, a kašnjenje za OpenRAVE aplikacijom jer otprilike upravo tih 50 ms što je dovoljno dobro za prikaz pozicija.

5.1. Brzina rada simulacije

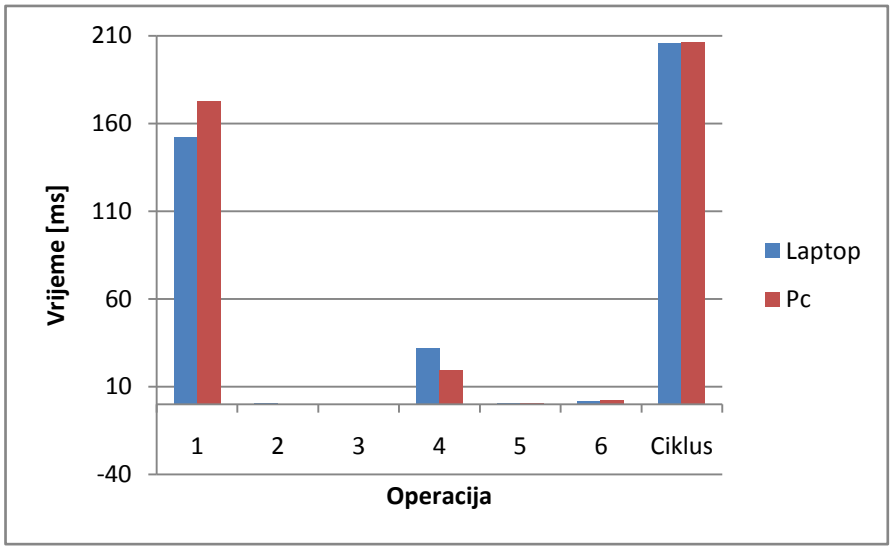
U programu su stavljeni i brojači vremena kojima se bilježi vrijeme nakon svakog koraka. Iz tih vremena je vidljivo potrebno vrijeme za svaki korak. U tablici je dana analiza tih vremena, gdje su prikazane minimalna i maksimalna vrijednost te prosječna vrijednost u kojoj je izvršen pojedini korak. Također je program isproban na PC-u u laboratoriju koji je s robotima povezan LAN mrežom i na laptopu koji je povezan pomoću Wi-Fi mreže s robotima. Prikaz podataka dan je u tablicama [Tablica 3 i Tablica 4].

	Min	Max	Sred
Slanje i primanje poruke TCP-om	0.116470	0.176165	0,152321
Obrada poruke s koordinatama	0.000182	0.000262	0,000209
Zamjena i provjera alata	0,000010	0,000012	0,000011
Gibanje robota	0.024104	0.065155	0,031766
Računanje vanjskih koordinata	0.000451	0.000557	0,000473
Spremanje poruke u txt za GUI	0.001432	0.001770	0,001623
Vrijeme ciklusa	0.152814	0,309770	0,205616

Tablica 3 . Vremena izvršavanja programa na laptopu

	Min	Max	Sred
Slanje i primanje poruke TCP-om	0,136952	0,188657	0,172817
Obrada poruke s koordinatama	0.000104	0.000157	0,000114
Zamjena i provjera alata	0,000001	0,000009	0,000005
Gibanje robota	0.015010	0.031389	0,019382
Računanje vanjskih koordinata	0.000231	0.000452	0,000239
Spremanje poruke u txt za GUI	0.001461	0.003307	0,002402
Vrijeme ciklusa	0.165850	0,226746	0,206194

Tablica 4. Vremena izvršavanja programa na PC-u



1. Komunikacija
2. Obrada poruke
3. Zamjena alata
4. Gibanje robota
5. Računanje vanjskih koordinata
6. Spremanje poruke

Slika 39. Prikaz vremena ciklusa

Iz priloženog grafa [Slika 39] vidljivo je da najviše vremena potrebnog za izvršavanje jednog koraka u programu otpada na komunikaciju računala i robota. U komunikaciji se Wi-Fi pokazao malo brži od LAN mreže. Pošto je PC snažniji od laptopa ima manja vremena izračunavanja matematičkih operacija, ali je to vrijeme u odnosu na ukupno vrijeme gotovo zanemarivo. Prednost PC-a je kod prikaza gibanja kretanja robota zbog snažnijeg procesora, te brže obrade slike. Konačno vrijeme procesa je gotovo jednako kod laptopa i PC-a. Iz prikazanih podataka ispada da je potrebno oko 200 ms za jedan ciklus.

Kod promjene alata ciklus traje nešto više jer je potrebno oko 120 ms za tu operaciju dok kad nema promjene alata ova operacija traje 5 μm. Kako robot u tom trenutku stoji jer mora čekati promjenu stanja ventila za zrak, te je prihvaćanje hvataljke zračno, ovaj ciklus je mnogo kraći nego ciklus realnog robota.

Vizualno gledano kretanje robota na PC zaslonu izgleda dosta vjerno i nisu vidljiva zastajanja.

5.2. Točnost pozicioniranja robota

Točnost pozicioniranja [Tablica 5] robota Fanuc LR Mate 200iC/5L u OpenRAVE-u je uspoređena s Fanucovim programom Roboguide. Usporedbom koordinata koje su u tablici dane za robota bez alata, može se zaključiti da OpenRAVE jako vjerno simulira stvarnog robota. Na jednake unutarnje koordinate, u oba programa vanjske koordinate su identične u tri decimale, gotovo u svim slučajevima. Najveće odstupanje je 5 μm kod pozicije, te 0,001° kod orijentacije. Uočeno je da točnost ovisi o području radnog prostora u kojem robot radi, te je u nekim dijelovima potpuno točno, a u nekim je primjećeno malo odstupanje.

Kod polja gdje je robot s mjernom kuglom vidljiva su mala odstupanja u orijentaciji alata. Do odstupanja je došlo iz razloga što stvarni alat ima zakret koordinatnog sustava u odnosu na prirubnicu robota od 179,003°oko X osi, -1,067° oko Y osi te -114,685° oko Z osi, dok alat koji je konstruiran za OpenRAVE ima zbog jednostavnijeg konstruiranja sljedeće zakrete. 180°oko X osi, 0°oko Y osi i -114,685°oko Z osi. Pomaci koordinatnog sustava su jednaki kod konstruiranog alata i stvarnog, stoga su koordinate pozicije točne kod OpenRAVE-a. Ako se u Roboguide-u stave pomaci centra alata kako je konstruiran alat za OpenRAVE, vrijednosti orijentacije su jednake. Iz toga se zaključuje da do greške nije došlo zbog OpenRAVE-a, nego zbog pojednostavljenja konstruiranja alata. Kad bi alat bio potpuno točno konstruiran nebi bilo odstupanja, osim što bi montaža na robota bila malo teža jer bi bilo potrebno rotirati alat prije spajanja.

1. ispitivanje												
J1	0,000	J2	0,000	J3	0,000	J4	0,000	J5	0,000	J6	0,000	deg
Roboguide bez alata						OpenRAVE bez alata						
X	565,000	Y	0,000	Z	475,000	X	565,000	Y	0,000	Z	475,000	mm
W	180,000	P	-90,000	R	0,000	W	180,000	P	-90,000	R	0,000	deg
Roboguide s mjernom kuglom						OpenRAVE s mjernom kuglom						
X	852,987	Y	67,865	Z	443,436	X	852,987	Y	67,865	Z	443,436	mm
W	-91,487	P	24,680	R	88,826	W	-90,000	P	24,685	R	90,000	deg
2. ispitivanje												
J1	-21,334	J2	24.162	J3	-28,621	J4	0,538	J5	-54,327	J6	-48,734	deg
Roboguide bez alata						OpenRAVE bez alata						
X	600,000	Y	-235,000	Z	155,000	X	600,000	Y	-234,995	Z	155,004	mm
W	-175,000	P	-5,000	R	-70,000	W	-175,000	P	-5,000	R	-70,000	deg
Roboguide s mjernom kuglom						OpenRAVE s mjernom kuglom						
X	684.737	Y	-196.757	Z	-127.658	X	684.737	Y	-196.751	Z	-127.653	mm
W	-7.629	P	-1.375	R	44.921	W	-6.636	P	-2.436	R	45.045	deg
3. ispitivanje												
J1	19,656	J2	-1,981	J3	-42,506	J4	106,276	J5	-120,000	J6	215,187	deg
Roboguide bez alata						OpenRAVE bez alata						
X	396,929	Y	71,157	Z	219,371	X	396,928	Y	71,157	Z	219,369	mm
W	-30,195	P	-53,281	R	131,807	W	-30,194	P	-53,282	R	131,805	deg
Roboguide s mjernom kuglom						OpenRAVE s mjernom kuglom						
X	496,509	Y	-169,482	Z	363,296	X	496,509	Y	-169,482	Z	363,296	mm
W	-122,230	P	2,976	R	2,782	W	-121,181	P	3,529	R	3,696	deg

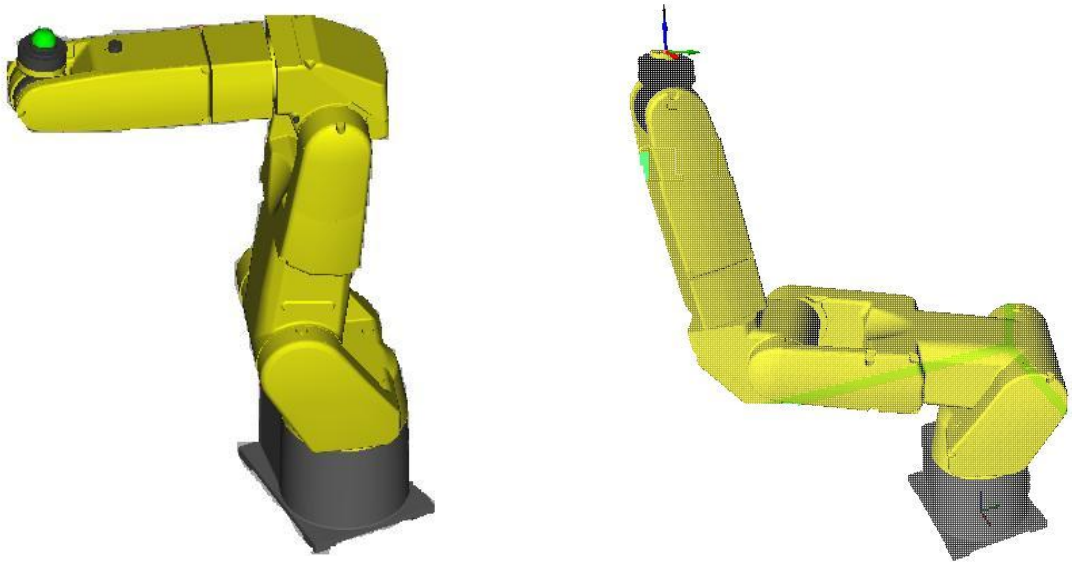
Tablica 5. Usporedba točnosti OpenRAVE-a upravljanog unutarnjim koordinatama

Kad je riječ o upravljanju vanjskim koordinatama [Tablica 6] situacije je malo drugačija. Ovdje dolazi do izražaja redundantnost robota [Slika 40] te OpenRAVE ponekad izabere poziciju koju robot u stvarnosti ne odabere. Što se tiče točnosti, ona je i dalje vrlo visoka i

odstupanja su svega nekoliko μm. Također, kada je odabrano redudantan položaj, robot u OpenRAVE-u je i dalje pravilno pozicioniran i orijentiran. Također i u ovom načinu upravljanja OpenRAVE pokazuje odlične rezultate.

1. ispitivanje											
Zadana pozicija			Roboguide			OpenRAVE					
X	748,399	mm	J1	-22,774	deg	J1	-22,774	deg	X	748,396	mm
Y	-244,080	mm	J2	45,114	deg	J2	45,114	deg	Y	-244,080	mm
Z	166,428	mm	J3	-30,929	deg	J3	-30,929	deg	Z	166,428	mm
W	66,554	deg	J4	55,375	deg	J4	55,375	deg	W	66,554	deg
P	16,350	deg	J5	79,135	deg	J5	79,135	deg	P	16,350	deg
R	121,236	deg	J6	57,939	deg	J6	57,938	deg	R	121,236	deg
2. ispitivanje											
Zadana pozicija			Roboguide			OpenRAVE					
X	642,624	mm	J1	38,669	deg	J1	38,669	deg	X	642,625	mm
Y	544,444	mm	J2	43,950	deg	J2	43,950	deg	Y	544,443	mm
Z	349,968	mm	J3	-3,444	deg	J3	-3,443	deg	Z	349,972	mm
W	76,249	deg	J4	55,375	deg	J4	55,375	deg	W	76,248	deg
P	51,835	deg	J5	20,969	deg	J5	20,969	deg	P	51,835	deg
R	135,098	deg	J6	86,502	deg	J6	86,501	deg	R	135,097	deg
3. ispitivanje											
Zadana pozicija			Roboguide			OpenRAVE					
X	-143,747	mm	J1	-108,548	deg	J1	71,452	deg	X	-143,745	mm
Y	-402,980	mm	J2	-6,983	deg	J2	-78,880	deg	Y	-402,978	mm
Z	556,130	mm	J3	0,682	deg	J3	96,073	deg	Z	556,129	mm
W	-7,763	deg	J4	-5,826	deg	J4	153,190	deg	W	-7,763	deg
P	-1,991	deg	J5	94,825	deg	J5	12,959	deg	P	-1,991	deg
R	13,674	deg	J6	57,702	deg	J6	84,413	deg	R	13,674	deg

Tablica 6. Usporedba točnosti OpenRAVE-a upravljanog vanjskim koordinatama



Slika 40. Redundantnost robota iz 3. Ispitivanja iz tablice [Tablica 6], lijevo je model u Roboguide-u, desno je OpenRAVE

Za ispitivanje linearnog gibanja robotu je zadano da se iz početne točke pomakne po liniji u novu točku koja je od početne udaljena -200 mm po X osi, 750 mm po Y osi i 50 mm po Z osi. Vanjske i unutarnje koordinate početne i završne pozicije dane su u tablici [Tablica 7]. Početna pozicija koji je robot trebao zauzeti je bila 500, -250, 150, te ovdje postoji malo odstupanje, ali nebitno za ovo ispitivanje jer se robot kreće u odnosu na ovu poziciju. Iz rezultata je vidljivo. Također treba napomenuti, kako je računanje operacije trajalo 1,5 min, pošto je u ovom primjeru stavljena rezolucija od 0,01 mm. Kada se rezolucija smanjila na 0,1 mm, računanje je trajalo 15 s, ali se smanjila točnost pozicioniranja. Sada je odstupanje iznosilo 0,02 mm. Treba napomenuti kako je usprkos pogrešci pozicioniranja, orijentacija uvijek ostala nepromijenjena.

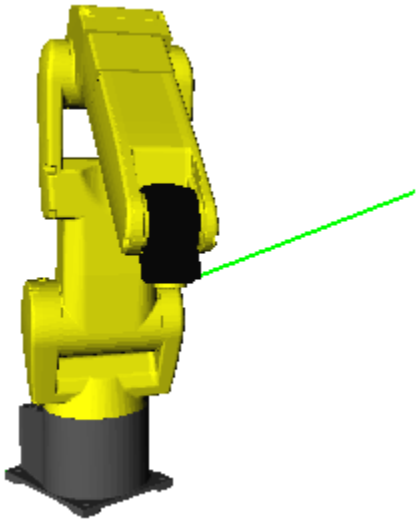
Početna točka						Završna točka					
J1	-26,595	mm	X	499,999	deg	J1	59,999	mm	X	300,001	deg
J2	12,930	mm	Y	-250,000	deg	J2	15,106	mm	Y	499,993	deg
J3	-33,010	mm	Z	149,996	deg	J3	-25,212	mm	Z	199,996	deg
J4	-0,273	deg	W	-175,001	deg	J4	7,811	deg	W	-175,001	deg
J5	-49,927	deg	P	-5,000	deg	J5	-64,365	deg	P	-5,000	deg
J6	-43,023	deg	R	-69,999	deg	J6	-133,138	deg	R	-69,999	deg

Tablica 7. Linearno gibanje robota

Drugo testiranje linearnog gibanja robota je takvo da mu zadamo koordinate sljedeće točke u koju se mora pomaknuti, tako da gibanje između trenutne i te nove bude linearno. Kod tog ispitivanja postignuti su sljedeći rezultati [Tablica 8]. Iz rezultata se vidi da je pozicioniranje prilično točno, te su odstupanja u granicama točnosti robota.

	Početna točka	Zadana točka	Postignuta točka	
X	500,000	300,000	300,004	mm
Y	-250,000	500,000	499,983	mm
Z	150,000	200,000	199,999	mm
W	-175,000		-175,000	deg
P	-5,000		-5,000	deg
R	-70,000		-70,000	deg

Tablica 8. Linearno gibanje robota



Slika 41. Iscrtana putanja

6. ZAKLJUČAK

OpenRAVE je programski paket koji pruža podršku za simuliranje robotskih sustava prije njihove implementacije u stvarnom okruženju. OpenRAVE je modul za Python tako da je korištenje prilično jednostavno ako se poznaje Python jezik.

Kod konstruiranja robota najlakše je kada se konstruira CAD model te se svi koordinatni sustavi postave na odgovarajuća mjesta. Ako nije tako, manipuliranje objektima je malo komplicirano pošto OpenRAVE sa svim objektima manipulira u odnosu na svoj koordinatni sustav.

Simuliranje kretnji robota po osima je dobro izvedeno i odvija se veoma brzo. Malo manje kvalitetno je gibanje robota pomoću vanjskih koordinata. OpenRAVE ima ugrađen Ikfast, koji omogućuje preračunavanje vanjskih koordinata u unutarnje ovisno o konfiguraciji robota. Taj posao obavlja odlično, što se tiče točnosti i brzine. Jedini nedostatak je što često odabere redundantnu konfiguraciju robota.

Kod linearnog gibanja računanje je prilično sporo ako se zahtijeva točnost na razini prethodnih načina gibanja.

Sama točnost pozicioniranja je odlična i odstupanje je u granicama točnosti robota.

OpenRAVE je odličan alat prvenstveno jer je besplatan i još nije završen, nego se razvija kako se naiđe na neki problem. OpenRAVE ima još mnoge mogućnosti koje nisu ovdje prezentirane te bi gotovo mogao zamijeniti alate za simuliranje koji dolaze uz robote.

PRILOZI

I. Programski Python kod

II. CD-R disk

LITERATURA

[1] Šimunović, Vladimir J.: Neurokirurgija, Medicinska naklada, Zagreb 2008

[2] Službene stranice Fanuc robot
http://www.fanucrobotics.com/, pristupljeno prosinac 2012.

[3] Službene stranice Kuka Robot Group
http://www.kuka-robotics.com/usa/en/, pristupljeno prosinac 2012.

[4] Službene stranice Deutches Zentrum für Luft- und Raumfahrt
http://www.dlr.de/dlr/en/ , pristupljeno siječanj 2013.

[5] Diankov, Rosen: OpenRAVE: A Planning Architecture for Autonomous Robotics, Carnegie Mellon University, 2008.
http://www.ri.cmu.edu/pub_files/pub4/diankov_rosen_2008_2/diankov_rosen_2008_2.pdf, pristupljeno siječanj 2013.

[6] OpenRAVE stranica
http://openrave.org/

[7] OpenRAVE forum
http://openrave-users-list.185357.n3.nabble.com/

[8] Python Manuals

PROGRAMSKI PYTHON KOD

```
from openravepy import *
from socket import *
import numpy, math, time
def joint(fanuc,Jf,kuka,Jk):                #potprogram kretanja robota
        #ulazni podatci su naziv robota i vektor kuteva zakreta
        manipfanuc = interfaces.BaseManipulation(fanuc)
        manipfanuc.MoveManipulator(goal=Jf)
        manipkuka = interfaces.BaseManipulation(kuka)
        manipkuka.MoveManipulator(goal=Jk)
        robot.WaitForController(0)
def fanuc_j(fan):       #obrada primljene poruke za fanuca
        k=fan.split( )   #rezanje stringa na mjestima praznog zraka
        alatf=k[7]
        k[3]=float(k[3])+float(k[2])            #korekcija treće osi zbog Fanuc upravljanja
        Jd=[float(k[1]),float(k[2]),float(k[3]),float(k[4]),float(k[5]),float(k[6])]
        Jr=numpy.radians(Jd)
        Jd[2]=Jd[2]-Jd[1]       #ponovno koregira treću koordinatu zbog ispisa
        return Jd,Jr,alatf
def world_f(fanuc):    #world koordinate kada je robot bez alata
        #vrača world koordinate robota uključujući korekciju u odnosu na openrave koordinatni sustav
        mht = fanuc.GetLinks()[6].GetTransform()
        #obrada mht
        X=mht[0,3]*1000
        Y=mht[1,3]*1000
        Z=mht[2,3]*1000
        eps=1e-10
        if abs(mht[0,0])<eps and abs(mht[1,0])<eps:
```

```
                R=0     #roll
                P=math.degrees(math.atan2(-mht[2,0],mht[0,0]))          #pitch
                W=math.degrees(math.atan2(-mht[1,2],mht[1,1]))          #yaw-roll
        else:
                Rr=math.atan2(mht[1,0],mht[0,0])
                sR=math.sin(Rr)
                cR=math.cos(Rr)
                R=math.degrees(Rr)
                P=math.degrees(math.atan2(-mht[2,0],mht[0,0]*cR+mht[1,0]*sR))
                W=math.degrees(math.atan2(mht[0,2]*sR-mht[1,2]*cR,mht[1,1]*cR-
mht[0,1]*sR))
        #korekcija koordinata zbog pomaka robota od koordinatnog sustava okruženja
        T=fanuc.GetTransform()      #mht pozicije robota
        T[2,3]+=0.330
        T[0:3,3]*=1000
        Wf=[X,Y,Z,W,P,R]
        Wf[0:3]-=T[0:3,3]
         #korekcija koordinata zbog rotacije robota oko Z osi za -90 deg
        x=Wf[0]
        Wf[0]=-Wf[1]
         Wf[1]=x
        Wf[5]=Wf[5]+90
        if Wf[5] > 180:
                preb=Wf[5]-180
                Wf[5]=-180+preb
        if Wf[5]<-180:
                 preb=-180-Wf[5]
                Wf[5]=180-preb
        return Wf       #vrača vektor s world koordinatama
def world_f1(fanuc,kugla):    #vrača world koordinatni sustav kada je priključen alat kugla
```

```
mht = kugla.GetLinks()[1].GetTransform()
X=mht[0,3]*1000
Y=mht[1,3]*1000
Z=mht[2,3]*1000
eps=1e-10
if abs(mht[0,0])<eps and abs(mht[1,0])<eps:
        R=0             #roll
        P=math.degrees(math.atan2(-mht[2,0],mht[0,0]))          #pitch
        W=math.degrees(math.atan2(-mht[1,2],mht[1,1]))          #yaw-roll
else:
        Rr=math.atan2(mht[1,0],mht[0,0])
        sR=math.sin(Rr)
        cR=math.cos(Rr)
        R=math.degrees(Rr)
        P=math.degrees(math.atan2(-mht[2,0],mht[0,0]*cR+mht[1,0]*sR))
        W=math.degrees(math.atan2(mht[0,2]*sR-mht[1,2]*cR,mht[1,1]*cR-
mht[0,1]*sR))
T=fanuc.GetTransform()
T[2,3]+=0.330
T[0:3,3]*=1000
Wf=[X,Y,Z,W,P,R]
Wf[0:3]-=T[0:3,3]
#korekcija koordinata zbog rotacije robota oko Z osi za -90 deg
x=Wf[0]
Wf[0]=-Wf[1]
Wf[1]=x
Wf[5]=Wf[5]+90
if Wf[5] > 180:
        preb=Wf[5]-180
        Wf[5]=-180+preb
```

```
        if Wf[5]<-180:
                preb=-180-Wf[5]
                Wf[5]=180-preb
        return Wf
def kuka_j(ku):         #obrada poruke za Kuka robota
        k=ku.split('x')
        alatk=k[8]
        Jd=[float(k[1]),float(k[2]),float(k[3]),float(k[4]),float(k[5]),float(k[6]),float(k[7])]
        Jr=numpy.radians(Jd)
        return Jd,Jr, alatk
def world_k(kuka):    #wrača world koordinate Kuke uključujući korekciju u odnosu
        #na openrave koordinatni sustav
        mht = kuka.GetLinks()[7].GetTransform()
        X=mht[0,3]*1000
        Y=mht[1,3]*1000
        Z=mht[2,3]*1000
        Rr=math.atan2(mht[1,0],mht[0,0])
        sR=math.sin(Rr)
        cR=math.cos(Rr)
        R=math.degrees(Rr)
        P=math.degrees(math.atan2(-mht[2,0],mht[0,0]*cR+mht[1,0]*sR))
        W=math.degrees(math.atan2(mht[0,2]*sR-mht[1,2]*cR,mht[1,1]*cR-mht[0,1]*sR))
        #korekcija zbog položaja robota u okruženju
        T=kuka.GetTransform()
        T[0:3,3]*=1000
        Wf=[X,Y,Z,W,P,R]
        Wf[0:3]-=T[0:3,3]
        return Wf
def pisanje(Jf,Wf,Jk,Wk):     #zapis poruke s koordinatama robota zbog ispisa u GUI
```

```
        #txt datoteke nije potrebno ručno izraditi
        koof=Jf+Wf  #stvara jedan vektor za fanuca
        kook=Jk+Wk
        out1=open('C:/Python26/fanuc.txt','w')  #otvaranje datoteke
        out1.write(str(koof))   #spremanje varijable
        out1.close()    #zatvaranje datoteke
        out2=open('C:/Python26/kuka.txt','w')
        out2.write(str(kook))
        out2.close()
def main():     #glavni program
        #kreiranje OpenRAVE okruženja
        env = Environment() #kreira okruženje
        env.SetViewer('qtcoin') #učitavanje GUI preglednika za OpenRAVE
        env.Load('data/fsb/laboratorij1.xml')   #učitavanje scene
#uspostavljanje komunikacije s robotima
        #kreiranje komunikacije s fanucom
        ADDRfanuc = ('192.168.123.25',1010)       #IP adresa i PORT robota
         clifanuc = socket( AF_INET,SOCK_STREAM)    #kreira konekciju
        clifanuc.connect((ADDRfanuc))       #spajanje na zadanu adresu
        #kreiranje komunikacije s kukom
        ADDRkuka = ('192.168.123.15',1111)
        clikuka = socket( AF_INET,SOCK_STREAM)
        clikuka.connect((ADDRkuka))
 #kreiranje robota i učitavanje alata
        fanuc=env.GetRobots()[0]              #postavljanje robota
        kuka=env.GetRobots()[1]
#definiranje početnih parametara za petlju
        alatfp=0        #početni alat fanuca -> 0 nema alata
        alatkp=0        #početni alat kuke -> 0 nema alata
```

```
        poruka='salji'   #poruka koja se šalje robotima
        i=0
        while 1:    #beskonačna petlja
#slanje poruke robotima
                clifanuc.sendall(poruka)
                clikuka.sendall(poruka)
#primanje poruke od robota
                fan = clifanuc.recv(4096)
                ku = clikuka.recv(4096)
                [Jdf,Jf,alatf]=fanuc_j(fan)
                [Jdk,Jk,alatk]=kuka_j(ku)
#promjena alata na fanucu
                # 0 -> nema alata
                # 1 -> mjerna kugla
                # 2 -> kamera s laserskim senzorom
                if int(alatf)==1:          #treba biti kugla
                        if alatfp==0:   #trenutno nema alata
                                T = fanuc.GetLinks()[7].GetTransform()      #koordinate
                        prirubnice
                                kugla=env.ReadRobotURI('robots/fsb/kugla.robot1.xml')
                                #učitavanje alata
                                 kugla.SetTransform(T)        #pomicanje
                                env.Add(kugla)          #učitavanje alata u okruženje
                                fanuc.Grab(kugla)      #fanuc prima alat, sada su spojeni
                                alatfp=1          #spremanje oznake koji je alat na robotu
                elif int(alatf)==2:               #treba laser
                        if alatfp==0:             #trenutno nema
                                T = fanuc.GetLinks()[6].GetTransform()
                                laser=env.ReadKinBodyXMLFile('robots/fsb/laser.xml')
                                laser.SetTransform(T)
```

```
                    env.Add(laser)
                    fanuc.Grab(laser)
                    alatfp=2
            elif int(alatf)==0:             #treba maknuti alat
                if alatfp==1:               #trenutno je kugla
                    fanuc.Release(kugla)        #fanuc otpušta alat
                    env.Remove(kugla)           #miče se alat iz okruženja
                    alatfp=0        #spremanje oznake da više nema alata
                elif alatfp==2:             #trenutno je laser
                    fanuc.Release(laser)
                    env.Remove(laser)
                    alatfp=0
#promjena alata na kuki
            # 0 -> nema alata
            # 1 -> kateter
            # 2 ->
            if int(alatk)==1:               #treba staviti kateter
                if alatkp==0:               trenutno je prazan
                    T = kuka.GetLinks()[kuka.GetActiveDOF()-1].GetTransform()
                    kateter=env.ReadKinBodyXMLFile('robots/fsb/kateter.xml')
                    kateter.SetTransform(T)
                    env.Add(kateter)
                    kuka.Grab(kateter)
                    alatkp=0
            elif int(alatk)==0:             #treba biti prazan
                if alatkp==1:               #trenutno je kateter
                    kuka.Release(kateter)
                    env.Remove(kateter)
                    alatkp=1
```

```
#gibanje robota
                joint(fanuc,Jf,kuka,Jk)          #pokretanje potprograma
#izračunavanje world koordinata fanuc
                if alatfp==0:              #ako nema alata
                        Wf=world_f(fanuc)
                elif alatfp==1:            #ako je kugla montirana
                        Wf=world_f1(fanuc,kugla)
                elif alatfp==2:            #ako je laser, zasad vrača koordinate prihvatnice
                        Wf=world_f(fanuc)
                #world koordinate kuka
                if alatkp==1:
                        Wk=world_k(kuka)
                elif alatkp==0:
                        Wk=world_k(kuka)
#spremanje koordinata u txt datoteku
                pisanje(Jdf,Wf,Jdk,Wk)
                i+=1
                if i==500:
                        break
#zatvaranje konekcija
        clifanuc.close()
        clikuka.close()
if __name__ == "__main__":
        main()
```